МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Реализация и исследование градиентных методов решения обратной задачи Штурма-Лиувилля на интервале»

Обучающийся И.С Никитин
(Инициалы Фамилия) (личная подпись)

Руководитель д.ф.-м.н., доцент, С.В. Талалов
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.пед.н., доцент, Т. С. Якушева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2022

Аннотация

Выпускная квалификационная работа посвящена реализации градиентного метода решения обратной задачи Штурма-Лиувилля на интервале и проведению исследований с готовой программой.

Ключевые слова: градиентный метод, обратная задача Штурма-Лиувилля, планарный световод, минимизация невязки.

В работе представлен алгоритм численного решения классической прямой задачи Штурма-Лиувилля методом стрельбы и алгоритм решения обратной задачи градиентным методом. Для исследования градиентного метода решения обратной задачи Штурма-Лиувилля бралось разное количество спектральных данных и различные начальные приближения функции, которую нужно восстановить. Численные эксперименты показывают, что при правильном подборе параметров возможно восстановить неизвестную функцию с заданной точностью.

Целью выпускной квалификационной работы является реализация градиентного метода решения обратной задачи Штурма-Лиувилля на интервале и проведение исследований с готовой программой.

Объектом исследования выпускной квалификационной работы является обратная задача Штурма-Лиувилля.

Предметом исследования выпускной квалификационной работы является градиентный метод решение обратной задачи Штурма-Лиувилля на интервале.

Выпускная квалификационная работа состоит из пояснительной записки на 51 страницу, включает в себя 18 рисунков, 4 таблиц, список из 25 источников, в том числе 5 источников на иностранном языке и два приложения.

Abstract

This graduation work is devoted to the implementation of the gradient method for solving the inverse Sturm-Liouville problem on the interval and to conducting research with a ready-made program.

Keywords: gradient method, inverse Sturm-Liouville problem, planar waveguides, residual minimization.

The graduation work presents an algorithm for the numerical solution of the classical direct Sturm-Liouville problem by the shooting method and an algorithm for solving the inverse problem by the gradient method. To investigate the gradient method for solving the inverse Sturm-Liouville problem, different amounts of spectral data and different initial approximations of the function to be recovered were taken. Numerical experiments show that, with the right choice of parameters, it is possible to restore the unknown function with a given accuracy.

The aim of the graduation work is to implement a gradient method for solving the inverse Sturm-Liouville problem on an interval and to conduct research with a ready-made program.

The object of the graduation work is the inverse Sturm-Liouville problem.

The subject of graduation work is a gradient method for solving the inverse Sturm-Liouville problem on the interval.

The graduation work consists of an explanatory note on 51 pages, includes 18 figures, 4 tables, the list of 25 references, including 4 foreign sources and 2 appendices.

Содержание

Введение

Цель выпускной квалификационной работы (ВКР) реализовать градиентный метод решения обратной задачи Штурма-Лиувилля на интервале и провести исследование с готовой программой. Для достижения цели ВКР необходимо выполнить следующие задачи:

– выбрать инструменты и технологии для программной реализации цели бакалаврской работы;

– реализовать решение прямой задачи Штурма-Лиувилля методом стрельбы для получения спектральных данных;

– реализовать градиентный метод решения обратной задачи Штурма-Лиувилля на интервале;

– провести тестирование программного продукта, решающего прямую и обратную задачу прямой Штурма-Лиувилля;

– провести численные эксперименты с готовой программой.

Объектом исследования выпускной квалификационной работы является обратная задача Штурма-Лиувилля.

Предметом исследования выпускной квалификационной работы является градиентный метод решение обратной задачи Штурма-Лиувилля на интервале.

Для понимания сути ВКР определим используемые понятия: обратные и некорректные задачи, градиентные методы, планарные волноводы, уравнение Шредингера.

Конкретного определения обратной задачи нету, это зависти от решаемой задачи. Прежде чем дать определение обратной задачи, нужно дать определение прямой задачи. Прямая задача – это нахождение следствий, обратная задача противоположна прямой задаче – это нахождение причин. В обратной задаче известны параметры, которые получились после решение прямой задачи. Решение обратной задачи — это нахождение того, что в прямой задаче известно и входит в условие прямой задачи. Чаще всего это

нахождение параметров модели при известных данных, которые получились в ходе эксперимента или наблюдений. Обратные задачи можно классифицировать: по искомой функции, по дополнительной информации, по уравнениям.

Задача называется корректной или корректно поставленной по Адамару если выполняются следующие условия: существования, единственности, устойчивости решения. Если хотя бы одно из трех условий не выполняется, то задача называется некорректной или некорректно поставленной [7]. Соответственно некорректной задачей называется задача, которая либо не имеет решения, либо имеет много решений, минимум два решения, либо решение неустойчиво, то есть при маленьком изменении входных параметров решение может сильно отличаться от точного решения.

Рассмотрим градиентные методы решения обратных и некорректных задач. Дано уравнение обратной задачи (1), записанной в операторном виде:

$$\mathrm{Aq} = \mathrm{f}, \tag{1}$$

где A: Q → F – оператор уравнения, дифференцируемый по Фреше;

Q и F – гильбертовы пространства.

В градиентном методе последовательно минимизируется функционал $\mathrm{J(q)} = \frac{1}{2}\|\mathrm{Aq} - \mathrm{f}\|^2 \to \min$. Для решения обратной задачи Штурма-Лиувилля в качестве функционала берется невязка, описывающая разницу между спектральными данными. Градиентный метод строит последовательность $\mathrm{q_0}, \mathrm{q_1}, \ldots, \mathrm{q_n}$ в которой следующее значение – это приближение искомого значения [14]. Последовательность строится по формуле $\mathrm{q_{n+1}} = \mathrm{q_n} - a \cdot \mathrm{J'q_n}$, где a – параметр, определяющий скорость градиентного спуска, $\mathrm{J'q_n}$ – градиент невязки. Элемент $\mathrm{J'q_n}$ указывает направление наискорейшего убывания функционала. Таким образом с помощью градиента невязки

последовательно минимизируется невязка. Последний элемент q_n – приближение искомого значения с заданной точность. Для сходимости градиентного метода необходимо ограничить функцию q.

Обратная задача Штурма-Лиувилля является задачей спектрального анализа, обратная классическим задачам о собственных значениях дифференциальных операторов. В классических задачах известен дифференциальный оператор и требуется найти спектральные характеристики(спектр) [20]. В спектральные характеристики могут входить: собственные числа, собственные функции и нормировочные коэффициенты. В обратной задаче Штурма-Лиувилля требуется найти дифференциальный оператор по его спектральным характеристикам. Пусть задано классическое уравнение Штурма-Лиувилля (2). Прямая задача состоит в нахождении нетривиальных решений уравнения на отрезке [a; b], с граничными условиями (3).

$$-y'' + q(x)y = \lambda y, \qquad x \in [0;\pi], \tag{2}$$

$$\begin{cases} \gamma_1 y'(a) + \beta_1 y(a) = 0, \ \gamma_1^2 + \beta_1^2 \neq 0 \\ \gamma_2 y'(b) + \beta_2 y(b) = 0, \ \gamma_2^2 + \beta_2^2 \neq 0' \end{cases} \tag{3}$$

где q(x) – потенциал;

γ, β – произвольные вещественные числа.

В результате решения этой задачи появляется набор из собственных чисел, собственных функций и нормировочных коэффициентов. Обратная задача Штурма-Лиувилля заключается в восстановлении потенциала q(x), имея данные рассеяния.

Теория обратных задач спектрального анализа активно развивается и имеет множество приложений. Такие задачи имеют приложения в таких сферах как физика, электроника, геофизика, метеорология и в других естественных науках. В квантовой механике уравнение Штурма-Лиувилля

называется стационарным уравнением Шредингера [8]. Стационарное уравнение Шредингера записывается следующим образом:

$$\frac{d^2\Psi(x)}{dx^2} + (E - u(x)\Psi(x) = 0, \quad (4)$$

где x – декартовая координата частицы;

Ψ – волновая функция;

$u(x)$ – потенциальная энергия частицы;

E – энергия частицы.

Решая уравнение (4) находят состояние частицы на различных уровнях энергии. Обратная задача состоит в отыскании потенциала $u(x)$ при известной энергии частицы и волновой функции Ψ [19]. В волноводной оптике распространение света может быть описано уравнением Гельмгольца (см. приложение А), которое можно свести к уравнению Штурма-Лиувилля. Решение прямой задачи – это волновые моды, которые образуются при определенном показателе преломления волноводного слоя. Обратная задача состоит в отыскании такого показателя преломления, который приводил бы к заданному набору волновых мод.

1 Формулировка математической модели

1.1 Обзор литературы

Теория обратных и некорректных задач стала развиваться в 50-х – 60-х годах XX века. С развитием теории стало понятно, что к обратным и некорректным можно отнести существенную часть задач. Активное развитие теории обусловлено увеличением скорости вычисления современной техники и увеличением количества областей, в которых применима данная теория. Существенный вклад в теорию обратных и некорректных задач внесли советские математики: А.Н Тихоново, В.К Иванова и М.М. Лаврентьев.

Опубликовано множество работ, связанных с решением обратной задачи Штурма-Лиувилля. В книге [7] собраны все основные направления теории обратных и некорректных задач, описаны методы решения и определены термины и теоремы, относящиеся к данным задачам, даются ответы на следующие вопросы: в каких разделах науки решаются обратные и некорректные задачи? суть некорректности и методы ее преодоления? можно ли построить устойчивые методы решения неустойчивых задач? В первой главе книги даны определения и примеры обратных и некорректных задач, во второй главе описаны градиентные методы решения некорректных задач, в третьей главе описан метод регуляризации с принципом выбора параметра регуляризации, в шестой главе описаны спектральные обратные задачи и обратные задачи рассеяния, в приложении прилагается обширная математическая теория, собранная для решения обратных и некорректных задач. Работа [18] посвящена решению некорректно поставленных математических задач. В своей книге Тихонов решает такие задачи приближено, опираясь не только на входные данные, но и на точность, с которой эти данные заданы. В данной работе даны определения корректных и некорректных задач, в настоящее время в математической литературе для задач с дополнительными условиями использует понятие «задачи, корректные по Тихонову». Тихонов разработал новый подход к решению

некорректно поставленных задач и назвал его методом регуляризации. Данный метод позволяет строить приближенное решение, устойчивое к малым изменениям исходных данных, для существенно некорректных задач. Так же в книге описываются уже существующие методы приближенного решения некорректных задач методом подбора и методом приближенного нахождения квазирешения. В книге [24] описываются различные методы численного решения уравнений Штурма-Лиувилля и Шредингера. Обсуждается реализация этих методов в программе Matlab, так же вместе с книгой прилагаются готовые пакеты Matlab и программа с графическим интерфейсом для решения уравнений Штурма-Лиувилля. В учебном пособии [9] классифицируются дифференциальные уравнения второго порядка и описываются основные методы решения краевых задач для дифференциальных уравнений второго порядка. Автор приводит примеры задач математической физики с описанием математической модели и с сопоставлением этой модели с физическими процессами. В книге [10] изложена теория обратных задач спектрального анализа. Автор для изложения современного состояния теории обратных задач спектрального анализа взял оператор Штурма-Лиувилля. В книге подробно описываются уже известные доказательства, теоремы и определения, связанные с задачами спектрального анализа, описывается теория операторов преобразования и конкретное применение ее к уравнению Штурма-Лиувилля. В книгах [10], [11] подробно описывается обратная задача квантовой теории рассеяния. В работе [4] автор дал определение прямой и обратной задачи рассеяния лазерного излучения в планарном волноводе, развил новый алгоритм решения обратной задачи волноводного рассеяния лазерного излучения, рассмотрел проблемы, связанные с обратными и некорректными задачами на примере рассеяния лазерного излучения в волноводе. В статье [21] описывается решения обратной задачи Штурма-Лиувилля методом наискорейшего спуска с использованием некоторых теорем для оптимизации алгоритма. Автор проводит исследование алгоритма на разных потенциалах с

использованием разного количества спектральных данных. Книга [1] посвящена решению некорректно поставленных задач методом итерационной регуляризации. В книге изложены обоснования регуляризирующих градиентных алгоритмов, в которых итерационный процесс прерывается при условии согласования погрешности данных с номером итерации. В учебном пособии [15] изложены численные методы решения обратных и некорректных задач для стационарных и нестационарных уравнений математической физики, дается исследование основных проблем, возникающих при приближенном решении обратных и некорректных задач, так же прилагается программная реализация и примеры расчетов. В работах [12], [2] приводятся результаты исследования регуляризирующих свойств метода простой итерации. В работах [2], [5] обоснован выбор параметра регуляризации по критерию невязки. В работах [21], [22], [25] описывается алгоритм решения обратной задачи Штурма-Лиувилля градиентным методом и проводятся исследования с разными потенциалами и начальными данными. В книге [23] дается элементарное и полное введение в теорию обратных задач, описываются основные идеи и методы теории обратных задач, приводятся различные применения обратных спектральных задач в естественных науках.

1.2 Формулировка математической модели в общем виде на примере планарного световода

Рассмотрим планарный волновод, показанный на рисунке 1. Он представляет из себя набор из трёх слоев: нижний – подложка с показателем преломлением n_2, верхний – покровный слой с показателем преломления n_3, середина – волноводный слой с показателем преломления n_1 и толщиной h. Свет распространяется в направлении оси z, а ось x перпендикулярна плоскости волновода. Световая волна, попадая в волноводный слой отражается от подложки и покровного слоя и остается в волноводном слое

при условии, если угол θ под которым она распространяется, удовлетворяется условию полного внутреннего отражения [3].

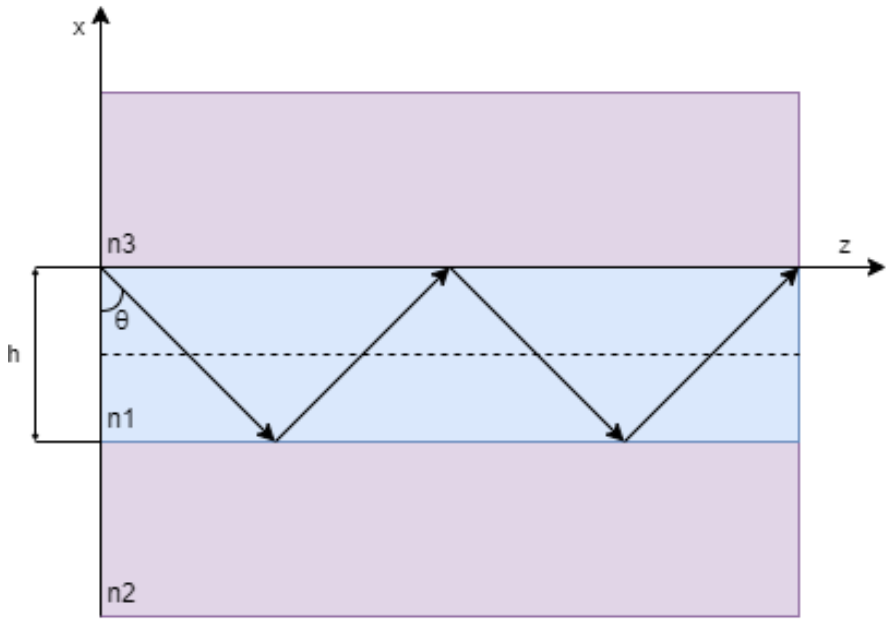

Рисунок 1 – Трехслойный планарный волновод

В волноводном слое направление распространения монохроматического света задается вектором $\vec{k}n_1$, где $\vec{k}$ – волновой вектор, n_1 – показатель преломления, $|\vec{k}| = k = \frac{2\pi}{\lambda} = \frac{\omega}{c}$, где ω – частота волны, λ – длина волны, c – скорость света в вакууме. Постоянная распространения световой волны $\beta = kn_1\sin(\theta)$.

«Так как световая волна, попадая в волноводный слой испытывает попеременное отражения от подложки и покровного слоя, то образуется стоячая волна, распространяющаяся вдоль оси z в направлении x. Стоячая волна может возникнуть только при определенных значения угла θ и постоянной распространения β. Стоячие волны, удовлетворяющие этим условиям, называются волновыми модами.» [6], [13]. Первые четыре волновые моды показаны на рисунке 2.

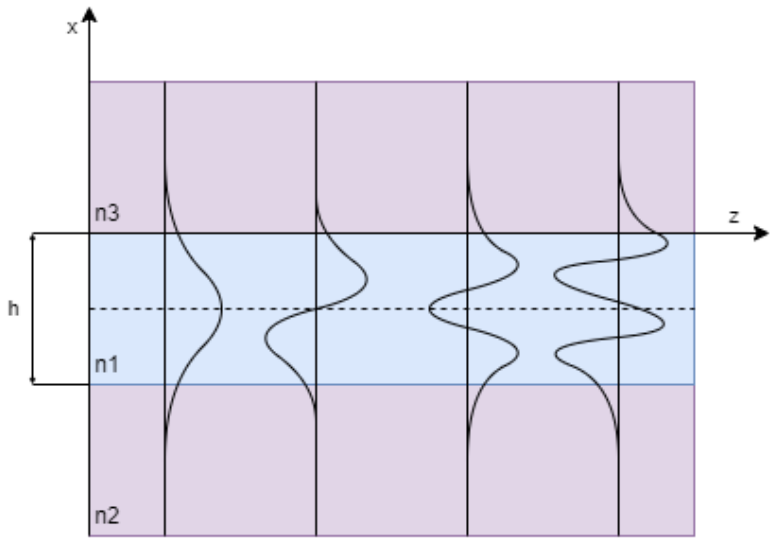

Рисунок 2 – Первые четыре волновые моды

Распространение излучения частоты ω может быть описано уравнением Гельмгольца:

$$(\Delta + \mathrm{k}^2 {n_1}^2)\mathrm{u} = 0, \tag{5}$$

где Δ – оператор Лапласа;

k – волновой вектор;

n_1 – показатель преломления.

Представим, что у нас имеется две координаты, тогда уравнение Гельмгольца записывается в таком виде:

$$\frac{\partial^2 \mathrm{u}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{u}}{\partial \mathrm{y}^2} + \mathrm{k}^2 {n_1}^2 \mathrm{u} = 0. \tag{6}$$

Пусть $\mathrm{u}(\mathrm{x}, \mathrm{y}) = \mathrm{X}(\mathrm{x})\mathrm{Y}(\mathrm{y})$, тогда уравнение (6) преобразуется к следующему виду:

$$\frac{X''}{X} + \frac{Y''}{Y} + k^2 {n_1}^2 = 0. \quad (7)$$

Пусть $\frac{X''}{X} = -\alpha$, тогда уравнение (7) преобразуется к виду:

$$\frac{Y''}{Y} + k^2 {n_1}^2 - \alpha = 0. \quad (8)$$

Перенесем коэффициент α в правую часть и умножим слагаемые на Y:

$$Y'' + k^2 {n_1}^2 = \alpha Y. \quad (9)$$

Если заменить: $Y'' \to -y''$, $k^2 {n_1}^2 \to q(x)$, $\alpha \to \lambda$, то получим уравнение Штурма-Лиувилля в виде уравнения (2).

Таким образом, с помощь разделения переменных мы свели уравнение Гельмгольца к уравнению Штурма-Лиувилля. Решение данного уравнение — это волновые моды, которые образуются при определенном показателе преломления волноводного слоя. Обратная задача состоит в отыскании такого показателя преломления, который приводил бы к заданному набору волновых мод.

1.3 Математическая модель прямой задачи Штурма-Лиувилля

Найти такие собственные числа λ, при которых существует нетривиальное решение уравнения (10), собственные функции Ψ должны удовлетворять граничным условиям (11).

$$-\frac{d^2\Psi(x)}{dx^2} + u(x)\Psi(x) = \lambda\Psi(x), \qquad x \in [0;\pi], \tag{10}$$

$$\begin{cases} \gamma_1 y'(a) + \beta_1 y(a) = 0, \ \gamma_1^2 + \beta_1^2 \neq 0 \\ \gamma_2 y'(b) + \beta_2 y(b) = 0, \ \gamma_2^2 + \beta_2^2 \neq 0 \end{cases}, \tag{11}$$

где $u(x)$ – функция непрерывная на $[0;\pi]$;

γ, β – произвольные вещественные числа.

В результате решения этой задачи получится набор из собственных чисел и собственных функций, чтобы получить полный спектр необходимо также найти нормировочный коэффициент α:

$$\alpha_n = \int_0^{\pi} \Psi^2(x)dx. \tag{12}$$

Данные рассеяния можно получить с помощью метода стрельбы. Метод стрельбы – вести пристрелку собственных чисел λ, меняя собственное число λ случайным или последовательным образом.

Рассмотрим алгоритм метода стрельбы [16]:

1. разбить отрезок [0;π] на M частей, определить шаг изменения собственного числа λ и записать этот шаг в переменную Δ,

2. заменить функцию $u(x)$ на интервале [0;π] на кусочно-постоянную функцию u^*,

3. для выбора числа M проконтролировать условие:

$$\frac{\|u(x) - u^*(x)\|}{\|u(x)\|} < \varepsilon, \tag{13}$$

где ε – относительная погрешность.

4. выбирать первое значение λ равное минимуму функции $u(x)$ на отрезке $[0; \pi]$,

5. запустить бесконечный цикл,

6. запустить итерационный цикл от 1 до M, с итерационным параметром j,

7. ввести обозначения $\Lambda_j = u_j + \lambda$,

8. записать решение на интервале:

при $\Lambda_j > 0$:

$$\Psi(x) = A \times \cos(\sqrt{\Lambda} * x) + B \times \sin(\sqrt{\Lambda} * x). \quad (14)$$

при $\Lambda_j < 0$:

$$\Psi(x) = A \times \cosh(\sqrt{\Lambda} * x) + B \times \sinh(\sqrt{\Lambda} * x). \quad (15)$$

9. если первая итерация в цикле($j = 1$), то вычислить константы A и B при условии:

$$\Psi(0) = 1, \quad (16)$$

$$\Psi'(0) = -Q_0, \quad (17)$$

где Q_0 – левое граничное условие.

10. если не первая итерация($j \neq 1$), то записать СЛАУ и найти константы A и B из СЛАУ:

при $\Lambda_{j-1} > 0$ и $\Lambda_j > 0$:

$$A_{j-1} \times \cos\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sin\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = A_j \times \cos\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sin\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (18)$$

$$-A_{j-1} \times \sqrt{\Lambda_{j-1}} \times \sin\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sqrt{\Lambda_{j-1}} \times \cos\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = -A_j \times \sqrt{\Lambda_j} \times \sin\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sqrt{\Lambda_j} \times \cos\left(\sqrt{\Lambda_j} \times x_{j-1}\right). \quad (19)$$

Решить систему линейных алгебраических уравнений из этого условия и найти константы A и B.

при $\Lambda_{j-1} > 0$ и $\Lambda_j < 0$:

$$A_{j-1} \times \cos\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sin\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = A_j \times \cosh\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sinh\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (20)$$

$$-A_{j-1} \times \sqrt{\Lambda_{j-1}} \times \sin\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sqrt{\Lambda_{j-1}} \times \cos\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = -A_j \times \sqrt{\Lambda_j} \times \sinh\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sqrt{\Lambda_j} \times \cosh\left(\sqrt{\Lambda_j} \times x_{j-1}\right). \quad (21)$$

Решить систему линейных алгебраических уравнений из этого условия и найти константы A и B.

при $\Lambda_{j-1} < 0$ и $\Lambda_j > 0$:

$$A_{j-1} \times \cosh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sinh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = A_j \times \cos\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sin\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (22)$$

$$-A_{j-1} \times \sqrt{\Lambda_{j-1}} \times \sinh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sqrt{\Lambda_{j-1}} \times \cosh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = -A_j \times \sqrt{\Lambda_j} \times \sin\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sqrt{\Lambda_j} \times \cos\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (23)$$

где Λ_j – параметр из 7 пункта.

Решить систему линейных алгебраических уравнений из этого условия и найти константы A и B.

при $\Lambda_{j-1} < 0$ и $\Lambda_j < 0$:

$$A_{j-1} \times \cosh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sinh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = A_j \times \cosh\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sinh\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (24)$$

$$-A_{j-1} \times \sqrt{\Lambda_{j-1}} \times \sinh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sqrt{\Lambda_{j-1}} \times \cosh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = -A_j \times \sqrt{\Lambda_j} \times \sinh\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sqrt{\Lambda_j} \times \cosh\left(\sqrt{\Lambda_j} \times x_{j-1}\right). \quad (25)$$

Решить систему линейных алгебраических уравнений из этого условия и найти константы A и B.

11. вычислить величину α:

$$\alpha(\lambda) = \int_{x_{j-1}}^{x_j} \Psi^2(x)dx. \quad (26)$$

12. увеличить итерационный параметр j на единицу и перейти к шагу 7,

13. после окончания цикла от 1 до M определить константы a и α:

$$a = \Psi'(\pi - 0) + Q_\pi(\pi - 0), \quad (27)$$

$$\alpha = \sum_{j=1}^{M} \alpha_j. \quad (28)$$

14. найти решения, если параметр a_{i-1} и a_i имеют разные знаки, то решение найдено на отрезке λ_{i-1} и λ_i,

15. проверить условие выхода из бесконечного цикла:

Если $\frac{\|N-\sqrt{\lambda_i}\|}{\|\sqrt{\lambda_i}\|} < \varepsilon$, то выйти из бесконечного цикла, N – количество найденных собственных чисел λ, λ_i – последнее найденное собственное число, ε – относительная погрешность, которая допускается при вычислении собственных чисел.

16. увеличить собственное число λ на шаг Δ, $\lambda = \lambda + \Delta$ и перейти к пункту 6.

Для прямой и обратной задачи Штурма-Лиувилля необходимо построить кусочно-постоянную функцию, аппроксимирующую значение заданной функции u(x) по норме с заданной точностью. Приведем алгоритм для построения кусочно-постоянной функции [16]. Пусть u(x) – кусочно-гладкая функция, $u^*(x)$ – кусочно-постоянная функция.

1) выбрать произвольное число $M \geq 1$, где M – число разбиений отрезка $[0; \pi]$,

2) разбить отрезок $[0; \pi]$ на M равных частей точками x_i, $i = 0 \dots M$, $x_0 = 0$, $x_M = \pi$,

3) выбрать середины отрезков – точки $\xi_j \in [x_{j-1}; x_j]$, $j = 0 \dots M$,

4) построить кусочно-постоянную функцию $u^*(x) = u(\xi_j)$, $j = 1 \dots M$,

5) вычислить величину $\delta = \|u(x) - u^*(x)\|$,

6) если δ больше относительной точности ε, то увеличить число разбиений отрезка $M \to M + 1$, перейти на шаг 2,

7) если $\delta \leq \varepsilon$, то закончить алгоритм.

Таким образом, описана математическая модель прямой задачи, приведен алгоритм метода стрельбы и алгоритм построения кусочно-постоянной функции, аппроксимирующую значение заданной функции.

1.4 Математическая модель обратной задачи Штурма-Лиувилля

Решения обратной задачи – это нахождение потенциала с помощью спектральных данных. Дано уравнение (29), найти функцию $u^*(x)$ такую, чтобы решение задачи приводило к набору собственных чисел ξ и нормировочному коэффициенту β. Решить данную задачу методом подбора с использованием градиентных методов.

$$-\frac{d^2\Psi(x)}{dx^2} + u^*(x)\Psi(x) = \lambda\Psi(x), \qquad x \in [0;\pi], \qquad (29)$$

где $u^*(x)$ – искомая функция;

λ – собственные числа.

Имеется набор спектральных данных $(\xi_1, \beta_1, \dots, \xi_n, \beta_n)$, используя данный набор нужно найти неизвестную исходную функцию $u^*(x)$. Использую априорную информацию подберем такую функцию $u(x)$, чтобы она могла приблизиться к первоначальной функции $u^*(x)$. Для подобранной функции $u(x)$ решить прямую задачу Штурма-Лиувилля. После этого имеется два набора спектральных данных:

– для неизвестной функции $u^*(x) - (\xi_1, \beta_1, \dots, \xi_n, \beta_n)$,

– для подобранной функции $u(x) - (\lambda_1, \alpha_1, \dots \lambda_n, \alpha_n)$.

Разность между двумя спектральными данными называется невязкой. Необходимо минимизировать различие между двумя спектральными данными и скорректировать подобранную функцию $u(x)$, чтобы она приближалась к неизвестной функции $u^*(x)$. Функция $u(x)$ корректируется с помощью градиента невязки. Метод подбора с использованием градиентных методов минимизирует невязку и корректирует подобранную функцию $u(x)$.

Приведем алгоритм градиентного метода решения обратной задачи Штурма-Лиувилля на интервале [16].

1) найти линейную часть приращения невязки [17]:

$$\Delta(u+\delta u)-\Delta(u)=\frac{1}{2}\left[\sum_{k=1}^{N}(\lambda_k+\delta\lambda_k-\xi_k)^2+\sum_{m=1}^{N}(\alpha_m-\beta_m)^2\right]-\frac{1}{2}\left[\sum_{k=1}^{N}(\lambda_k-\xi_k)^2+\sum_{m=1}^{N}(\alpha_m-\beta_m)^2\right]\approx\sum_{k=1}^{N}(\lambda_k-\xi_k)\times\delta\lambda_k\approx\sum_{k=1}^{N}\frac{1}{\alpha_k}(\lambda_k-\xi_k)\times(\Psi_k,\delta u\Psi_k), \quad (30)$$

где N – количество спектральных данных;

δu – приращение аргумента;

λ_k – собственные числа подобранной функции;

ξ_k – собственные числа искомой функции;

Ψ_k – собственные функции подобранной функции;

α_k – нормировочный коэффициент подобранной функции.

2) используя пункт 1 найти в общем виде компоненты градиент невязки g:

$$g_m=\sum_{k=1}^{N}\frac{1}{\alpha_k}(\lambda_k-\xi_k)\times(\Psi_k,\chi_m\Psi_k), \quad (31)$$

где χ характеристическая функция отрезка $[x_{i-1};x_i]$;

λ_k – собственные числа подобранной функции;

ξ_k – собственные числа искомой функции;

Ψ_k – собственные функции подобранной функции;

α_k – нормировочный коэффициент подобранной функции.

3) найти градиент для всех отрезков функции u(x),

4) скорректировать подобранную функцию $u_0(x)$:

$$u_0(x) \to u_1(x) = u_0 + \delta u_0, \tag{32}$$

$$\delta u_0 = -\varepsilon \sum_{m=1}^{M} g_m \times \chi_m(x). \tag{33}$$

5) скорректировать собственные числа λ:

$$\lambda_i \to \lambda_i + \delta\lambda_i, \tag{34}$$

$$\delta\lambda_i = \frac{1}{\alpha_i}(\Psi_i, \delta u \Psi_i). \tag{35}$$

6) скорректировать нормировочный коэффициент α:

$$\alpha_i \to \alpha_i + \sum_{k \neq i} \alpha_k \times |\eta_{ik}|^2, \tag{36}$$

$$\eta_{ik} = \frac{(\Psi_m, u(x) \times \Psi_i)}{\alpha_k(\lambda_i - \lambda_k)}. \tag{37}$$

7) вычислить невязку скорректированной функции u(x), используя исправленные λ и α:

$$\Delta\,(u) = \frac{1}{2}\left(\sum_{k=1}^{N}(\lambda_k - \xi_k)^2 + \sum_{m=1}^{N}(\alpha_m - \beta_m)^2\right). \tag{38}$$

8) сравнить значение невязки с величиной δ, если $\Delta\,(u) < \delta$, то выйти из цикла.

Таким образом, описана математическая модель обратной задачи и приведен алгоритм градиентного метода решения обратной задачи Штурма-Лиувилля на интервале.

1.5 Вывод по первому разделу

В первом разделе приведена вся основная литература, связанная с решением задач выпускной квалификационной работы, сформулирована математическая модель в общем виде на примере планарного волновода, показана связь между математической модели планарного волновода и задачи Штурма-Лиувилля на интервале. В разделе описана математическая модель прямой и обратной задачи Штурма-Лиувилля на интервале, приведен алгоритм метода стрельбы для решения прямой задачи Штурма-Лиувилля и приведен алгоритм градиентного метода для решения обратной задачи Штурма-Лиувилля.

2 Реализация алгоритма решения обратной задачи Штурма-Лиувилля

2.1 Методы исследования

Вычисления проводились на 64 битном процессоре AMD Ryzen 5 2500U with Radeon Vega Gfx 2.00 GHz, 8 ГБ ОЗУ, в операционной системе Windows 10, в среде разработки Visual Studio Code, на языке Python версии 3.10.0. Для разработки программы необходимы следующие библиотеки языка программирования python:

– Math - встроенная библиотека, предоставляющая доступ к математическим функциям и константам;

– Numpy - библиотека с открытым исходным кодом. Этот пакет предназначен для научных вычислений в языке Python. Библиотека предоставляет огромный выбор типов данных для удобства вычислений и программирования, такие как многомерные массивы, маскированные массивы, матрицы, вектора. Предоставляет набор подпрограмм для быстрых вычислений с использованием различных типов данных, позволяет делать логические операции, изменять тип данных, сортировать, делать преобразование Фурье, делать операции линейной алгебры, базовой статистики и много другое;

– Matplotlib - это библиотека позволяет делать статические, анимированные и интерактивные изображения в языке Python;

– Scipy - библиотека с отрытым исходным кодом, предоставляет фундаментальные алгоритмы для научных вычислений в языке Python;

– Sympy - библиотека для символьной математики. Включает в себя функции символьной арифметики, дискретной математики, математического анализа и алгебры.

Таким образом, описана среде в которой проводились исследования и описаны все необходимые библиотеки языка Python для разработки программы.

2.2 Аппроксимация исходной функции

Реализуем алгоритм для аппроксимации исходной функции u(x). В начале программы зададим относительную точность пользовательским вводом и первоначальное разбиение отрезков M. Зададим цикл с условием - если норма станет меньше относительной точности, то выйти из цикла. В цикле разбиваем отрезок [0; π] на M частей. Вычисляем евклидову норму, так как на каждой итерации мы увеличиваем число разбиений отрезков, то расстояние между исходной и аппроксимирующей функциями уменьшается. Выходим из цикла, когда норма станет меньше относительной точности. С помощью функции step() из библиотеки matplotlib строим ступенчатую функцию и отображаем на графике. Алгоритм аппроксимации приведен на рисунке 3, полный код программы описан в приложении Б.

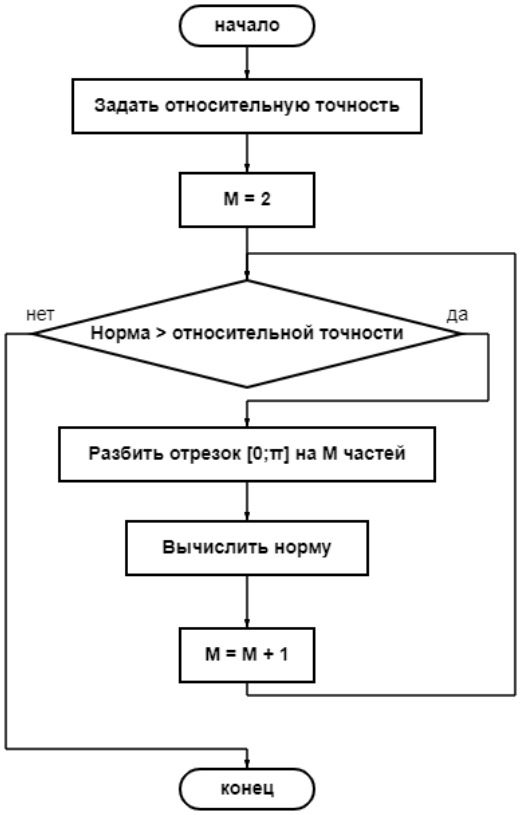

Рисунок 3 – Алгоритм аппроксимации исходной функции.

Приведем фрагмент кода программы, в котором в цикле отрезок $[0; \pi]$ разбивается на части и вычисляется норма.

```
while delta > e:
    kolIter += 1

# делим отрезок [a:b] на m частей
    xx = np.linspace(a, b, m)
# вычисляем расстояние между соседними точками

    fx = y(xx)
    xjminus1 = xx[0]
    sum1 = 0
#   находим
    sum2, err = integrate.quad(f2, a, b)

    for x in xx[1:]:
        # вычисляем середины соседних отрезков
        Ej = (xjminus1 + x)/2
# v1 для вычисления нормы
        v1, err = integrate.quad(f1, xjminus1, x)
        sum1 += v1
        xjminus1 = x
# вычисляем величину дельта, для условия выхода из цикла
    delta1 = pow(sum1, 1/2)
    delta2 = pow(sum2, 1/2)
    delta = de
```

Зададим параболу на отрезке $[0; \pi]$ с относительной точностью 15%, результат программы представлен на рисунке 4, число разбиений отрезков M рано 10.

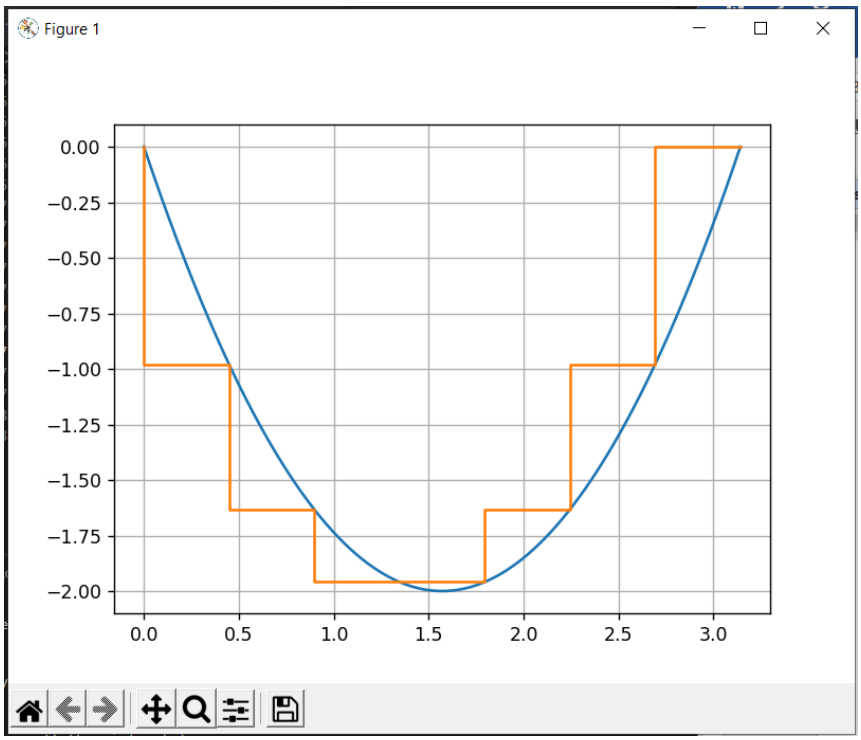

Рисунок 4 – Аппроксимация исходной функции

Таким образом по заданной точности функция разбивается на M отрезков и строится аппроксимация.

2.3 Реализация прямой задачи Штурма-Лиувилля на интервале

Приведем блок схему метода стрельбы на рисунке 5. В начале блок схемы выполняется аппроксимация исходной функции, далее определяются массивы, которые будут хранить данные рассеяния, задается точность решения, от которого зависит количество дынных рассеяния. Задается собственное значение λ, которое увеличивается от значения минимума исходной функции до значения, при котором сработает условие выхода из цикла. Задается бесконечный цикл, а внутри него итерационный цикл с параметром j от 1 до M, где M – число разбиений ступенчатой функции. В цикле от 1 до M в зависимости от знака переменной Λ выбирается собственная функция, если это первая итерация, то константы A и B находятся из условия уравнения 14, 15, иначе происходит сшивка решения и

константы A и B находятся из решения системы линейных алгебраических уравнений согласно пункту 10 алгоритма метода стрельбы. После итерационного цикла проверяется – имеется ли решение при текущем собственном значении? если решение есть, то решение ищется подробно между последними собственными числами и сохраняется в массив решений. Проверяется условие выхода, если оно выполняется, то выводятся все необходимые данные и алгоритм заканчивается, иначе собственное число увеличивается и решение ищется на следующем промежутке.

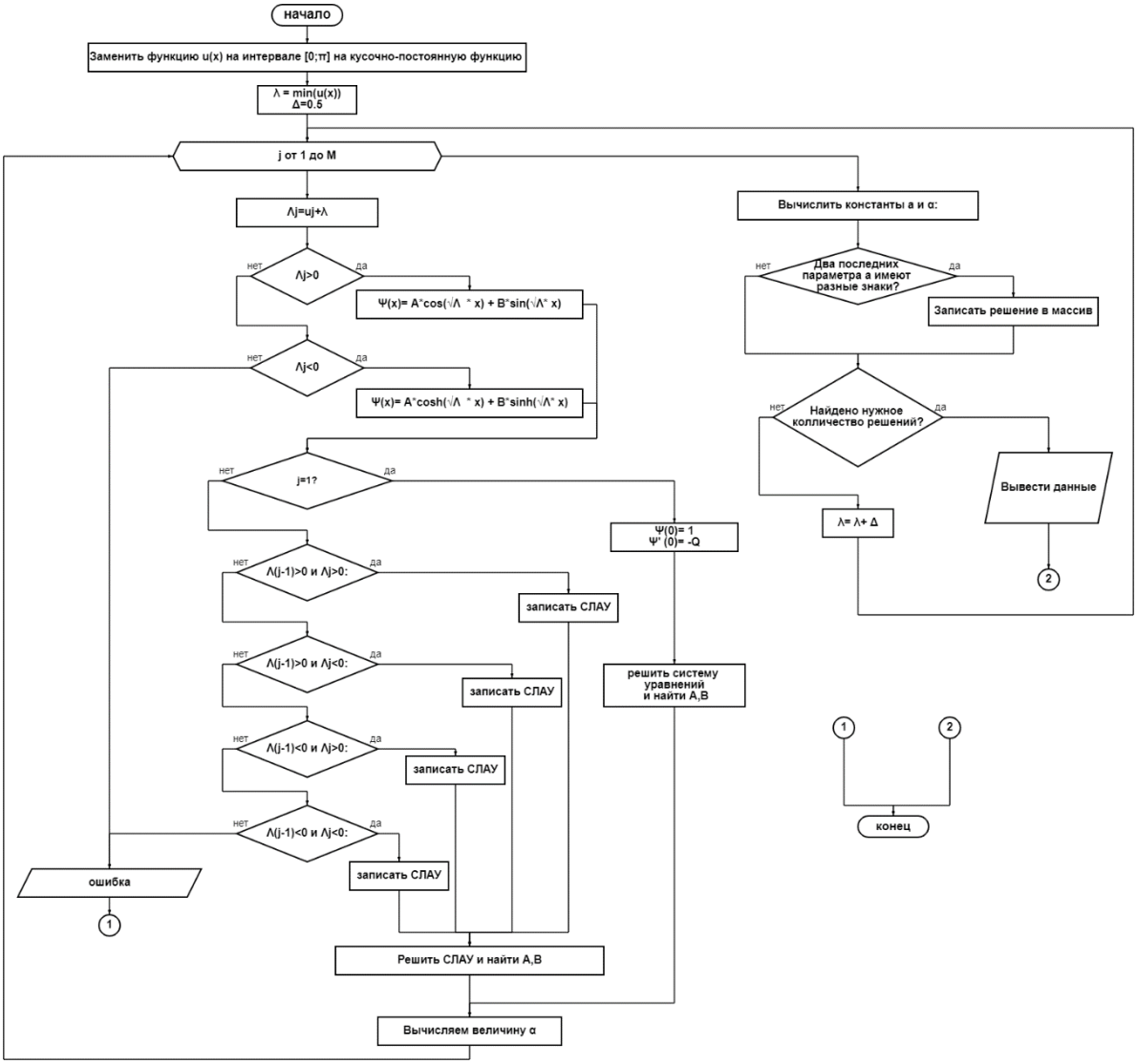

Рисунок 5 – Блок схема алгоритма метода стрельбы

Приведём фрагмент кода, определяющий собственную функцию. В зависимости от знака переменной Λ_j определяется собственная функция.

```
Λ = u[j] + λ
sqrt_Λ = np.sqrt(abs(Λ))
j_1 = j - 1
A, B, X = symbols('A, B, X')

if Λ > 0:
    funΨ = A * sym.cos(sqrt_Λ * X) + B * sym.sin(sqrt_Λ * X)
    def Ψ(x):
        return A * sym.cos(sqrt_Λ * x) + B * sym.sin(sqrt_Λ * x)
if Λ < 0:
    funΨ = A * sym.cosh(sqrt_Λ * X) + B * sym.sinh(sqrt_Λ * X)
    def Ψ(x):
        return A * sym.cosh(sqrt_Λ * x) + B * sym.sinh(sqrt_Λ * x)
```

Приведем фрагмент кода, определяющий параметры A и B. Если это первая итерация в цикле с параметром j от 1 до M, то коэффициенты A и B

```
if j == 1:
    A, B, X = symbols('A, B, X')

    funΨ = funΨ - 1
    funΨdx = sym.diff(funΨ, X)
    funΨdx = funΨdx + Q0

    funΨ = funΨ.subs(X, 0)
    funΨdx = funΨdx.subs(X, 0)

    A, B = list(sym.solve([funΨ,  funΨdx], A, B).values())
```

вычисляются следующим образом:

Этот фрагмент кода соответствует пункту 9 в алгоритме решения прямой задачи.

Если это не первая итерация, то вычисляем коэффициенты A и B из системы линейных алгебраических уравнений следующим образом:

```
else:
    sqrt_Λ_minus1 = np.sqrt(abs(array_Λ[-1]))
    A, B = symbols('A, B')

    if array_Λ[-1] > 0 and Λ > 0:
        A, B = list(sym.solve([array_A[-1] * sym.cos(sqrt_Λ_minus1 * x[j_1])
+ array_B[-1] * sym.sin(sqrt_Λ_minus1 * x[j_1]) - A * sym.cos(sqrt_Λ *
x[j_1]) - B * sym.sin(sqrt_Λ * x[j_1]),
                              -array_A[-1] * sqrt_Λ_minus1 *
sym.sin(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cos(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sin(sqrt_Λ * x[j_1]) - B
* sqrt_Λ * sym.cos(sqrt_Λ * x[j_1])], A, B).values())

    elif array_Λ[-1] > 0 and Λ < 0:
        A, B = list(sym.solve([array_A[-1] * sym.cos(sqrt_Λ_minus1 * x[j_1])
+ array_B[-1] * sym.sin(sqrt_Λ_minus1 * x[j_1]) - A * sym.cosh(sqrt_Λ *
x[j_1]) - B * sym.sinh(sqrt_Λ * x[j_1]),
                              -array_A[-1] * sqrt_Λ_minus1 *
sym.sin(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cos(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sinh(sqrt_Λ * x[j_1]) - B
* sqrt_Λ * sym.cosh(sqrt_Λ * x[j_1])], A, B).values())

    elif array_Λ[-1] < 0 and Λ > 0:
        A, B = list(sym.solve([array_A[-1] * sym.cosh(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sinh(sqrt_Λ_minus1 * x[j_1]) - A *
sym.cos(sqrt_Λ * x[j_1]) - B * sym.sin(sqrt_Λ * x[j_1]),
                              -array_A[-1] * sqrt_Λ_minus1 *
sym.sinh(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cosh(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sin(sqrt_Λ * x[j_1]) - B
* sqrt_Λ * sym.cos(sqrt_Λ * x[j_1])], A, B).values())

    elif array_Λ[-1] < 0 and Λ < 0:
        A, B = list(sym.solve([array_A[-1] * sym.cosh(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sinh(sqrt_Λ_minus1 * x[j_1]) - A *
sym.cosh(sqrt_Λ * x[j_1]) - B * sym.sinh(sqrt_Λ * x[j_1]),
                              -array_A[-1] * sqrt_Λ_minus1 *
sym.sinh(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cosh(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sinh(sqrt_Λ * x[j_1]) -
B * sqrt_Λ * sym.cosh(sqrt_Λ * x[j_1])], A, B).values())
```

Данный фрагмент соответствует пункту 10 в алгоритме «метод стрельбы».

```
def Ψ_2(x):
    return Ψ(x)**2

integr, err = integrate.quad(Ψ_2, x[j_1], x[j])
array_α.append(integr)
```

Определим нормировочный коэффициент α в итерационном цикле:

Данный код соответствует пункту 11 в алгоритме «метод стрельбы».

После итерационного цикла определим нормировочный коэффициент $\boldsymbol{\alpha}$

```
x_symbol = sym.Symbol('x')
Ψdx = sym.diff(Ψ(x_symbol))
a_ = float(Ψdx.subs(x_symbol, π) + Qπ * Ψ(π))
a.append(a_)

α_ = sum(array_α)
α.append(α_)
```

и параметр **a** на всем отрезке [0; π]:

Данный код соответствует пункту 13 в алгоритме «метод стрельбы».

Если два последних параметра $\boldsymbol{\alpha}$ имеют разные знаки, то решение найдено на отрезке $\boldsymbol{\lambda_{i-1}}$ и $\boldsymbol{\lambda_i}$. Приведем фрагмент кода, проверяющий

```
# Найден корень
if (len(a) >= 2):
    left = a[len(a)-2]
    right = a[len(a)-1]
    if sign(left) != sign(right):
        print("λ=", λ, "a=", a_, "α(λ)=", α_)
```

наличие решения.

Данный фрагмент кода соответствует пункту 14 в алгоритме «метод стрельбы».

Приведем фрагмент кода определяющий условие выхода из цикла. В этом фрагменте count_λ – количество найденных собственных чисел, λ_graph[−1] – последнее собственное число, eps_shoot - относительная погрешность, которая допускается при вычислении собственных чисел.

```
# Проверка точности
if count_λ > 0 and λ > 0:
    diff = abs(
        count_λ+1 - math.sqrt(abs(λ_graph[-1]))) / math.sqrt((abs(λ_graph[-1])))
    print("λ=", λ, " /a=", a_, " /α(λ)=", α_, " / условие выхода из цикла (",
          diff, "<=", eps_shoot, ") / количество λ count_λ=", count_λ)
    if diff <= eps_shoot:
        break
```

Данный фрагмент соответствует пункту 15 в алгоритме «метод стрельбы».

Если условие выхода из цикла не выполняется, то собственное число λ увеличивается и решение ищется в другой области.

В результате выполнения программы, решающей прямую задачу Штурма-Лиувилля на интервале [0; π], получаются следующие данные: набор из собственных чисел λ, собственных функций Ψ, нормировочных коэффициентов α. Приведем результат программы на примере параболы на отрезке $[0; \pi]$, с минимумом $y = -2$, с относительной точностью 20%. Набор из двенадцати собственных чисел и нормировочных коэффициентов приведен в таблице 1.

Таблица 1 – данные рассеяния

№	Собственное число (λ)	Нормировочный коэффициент (α)
1	-1.229	1.489
2	0.101	1.897
3	2.830	1.630
4	7.711	1.567
5	14.637	1.597
6	23.714	1.579
7	34.782	1.585
8	47.737	1.568
9	62.713	1.573
10	79.682	1.576
11	98.718	1.572
12	119.753	1.574

Приведем график зависимости параметра стрельбы от нормировочного коэффициента на рисунке 6. График представляет из себя волну с

увеличением длины и амплитуды. Найденные собственные числа отмечены красной точкой, видно, что с увеличением количества собственных чисел увеличивается разность между последними найденными собственными числами.

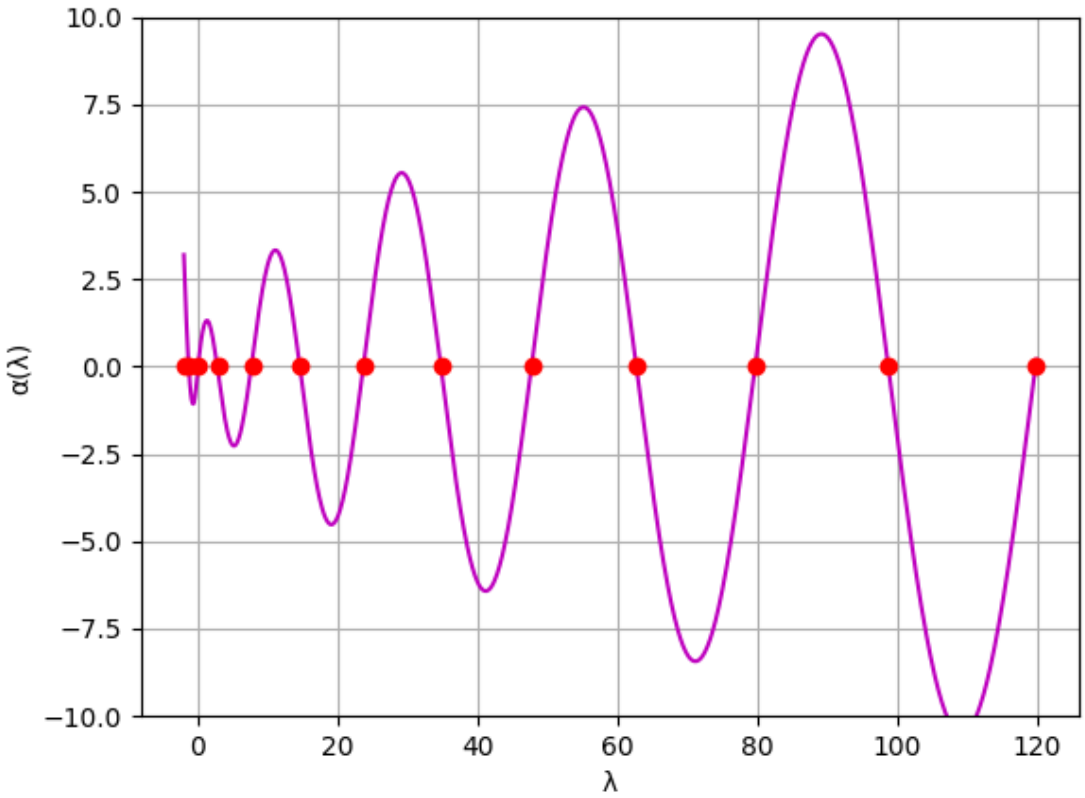

Рисунок 6 – График зависимости параметра стрельбы от нормировочного коэффициента

Приведем графики собственных функций на рисунках 7. По графикам видно, чем больше собственное число тем меньше переод и больше частота функции.

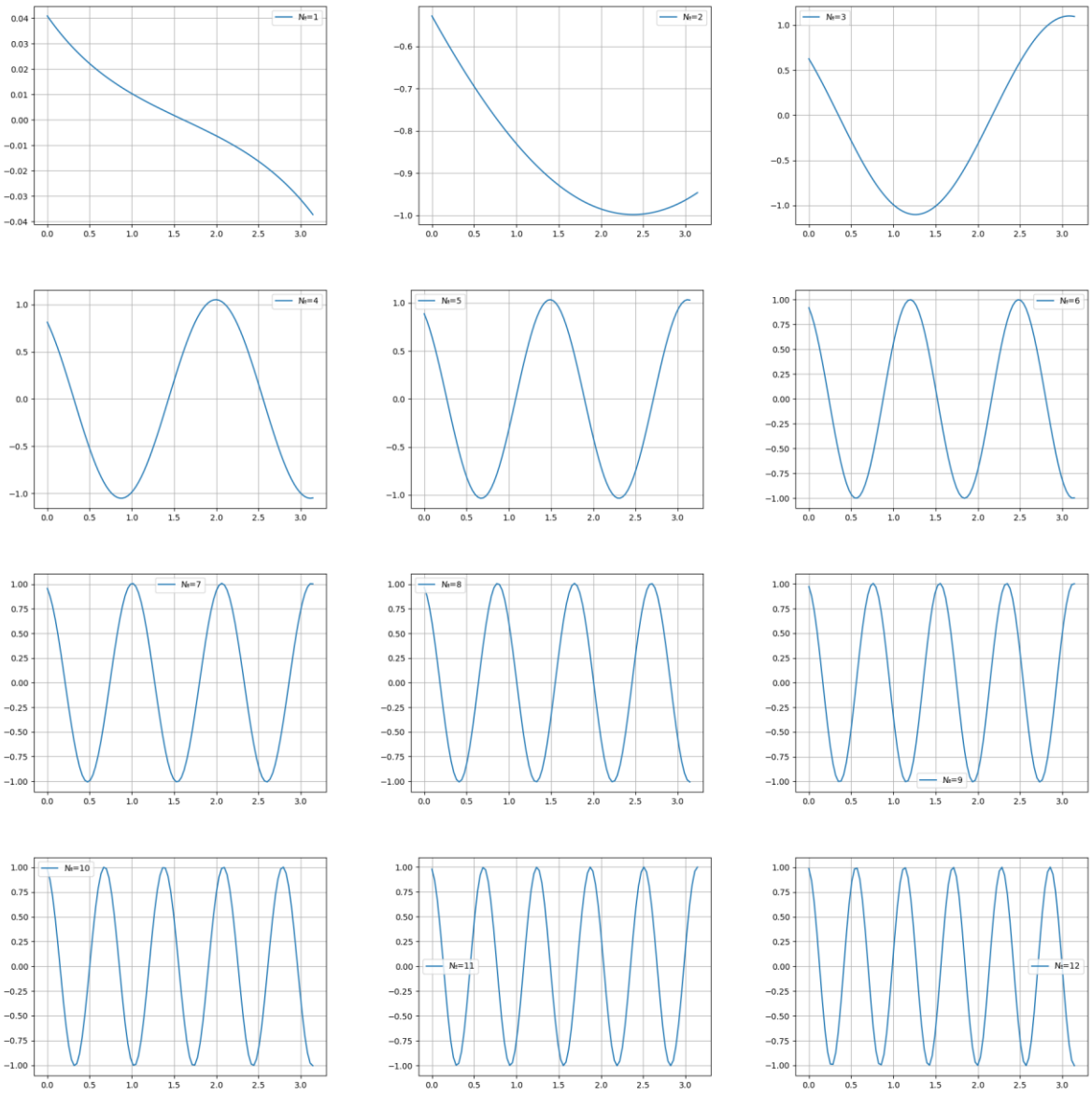

Рисунок 7 – Двенадцать найденных собственных функций

Таким образом реализован алгоритм решения прямой задачи Штурма-Лиувилля на интервале для получения собственных функции, собственных чисел и нормировочных коэффициентов.

2.4 Реализация обратной задачи Штурма-Лиувилля на интервале

Приведем реализацию алгоритма, решающего обратную задачу Штурма-Лиувилля. Исходные данные: набор спектральных данных $(\xi_1, \beta_1, \dots, \xi_n, \beta_n)$, полученный с использованием неизвестной функции $u^*(x)$ и подобранная функция $u(x)$. Используя программу, решающую прямую задачу Штурма-Лиувилля получим набор спектральных данных $(\lambda_1, \alpha_1, \dots, \lambda_n, \alpha_n)$ с использованием подобранной функции $u(x)$. Блок схема алгоритма решения обратной задачи приведена на рисунке 8. Имея набор спектральных данных исходной функции и набор спектральных данных подобранной функции можно скорректировать подобранную функцию и спектральные данные этой функции, для этого вычисляется градиент невязки согласно формуле (31) в алгоритме решения обратной задачи. На основе градиента невязки корректируется подобранная функция $u(x)$ и скорректированная функция строится на графике. Далее корректируется собственное число и нормировочный коэффициент согласно формулам (34) и (36) соответственно. Вычисляется невязка со скорректированными значениями. Если невязка больше относительной точности, то цикл продолжает корректировать подобранную функцию и данные рассеяния, иначе выводятся данные и алгоритм заканчивается.

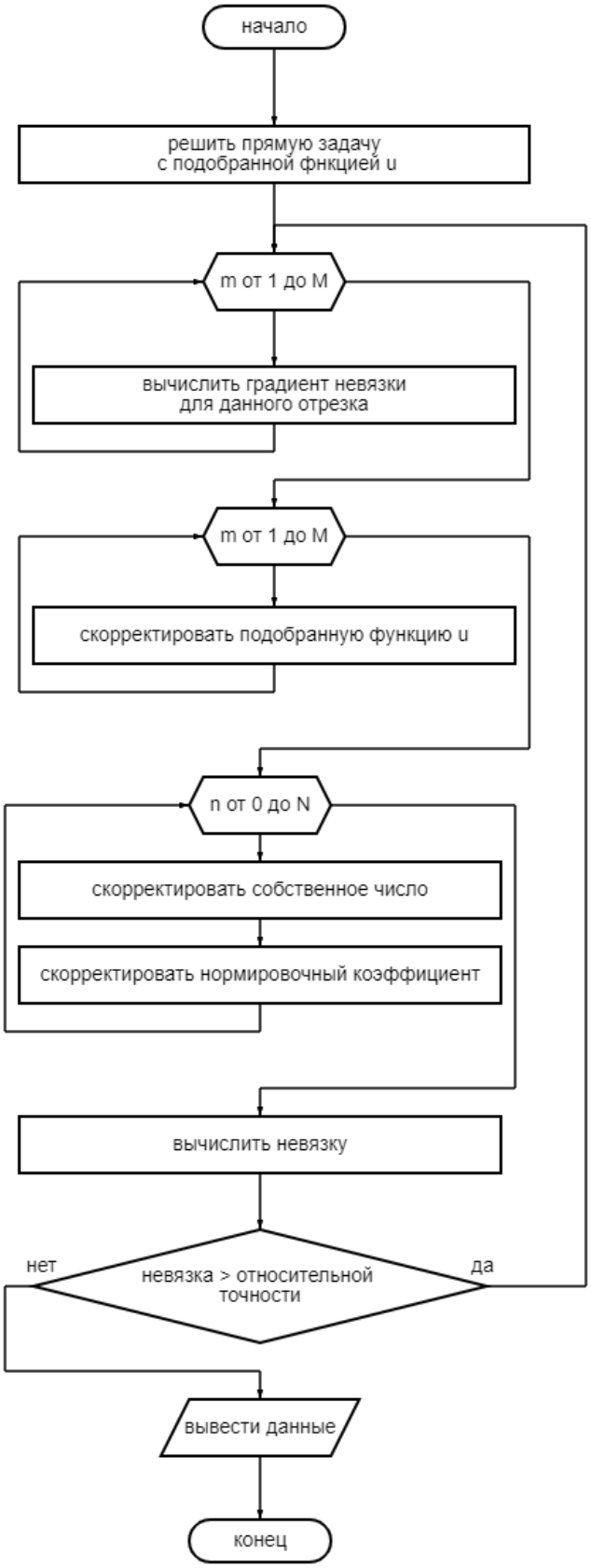

Рисунок 8 – Блок схема решения обратной задачи

Приведём код программы для вычисления градиента невязки. Во внешнем цикле задана функция Хэвисайда, во внутреннем цикле рассчитывается сумма согласно формуле (31) в пункте 2 решения обратной задачи. Каждому отрезку соответствует свой элемент вектора градиента невязки.

```
_g = []
for m in range(1, M):
    _sum = 0
    _rangeX = [arrayX[m-1], arrayX[m]]
    _left = _rangeX[0]
    _right = _rangeX[1]

    def _Xm(_x):
        if (_left  <= _x <= _right):
            return 1
        else:
            return 0

    for k in range(N):
        _Ψ = spectr_arr_2[k].Ψ
        _α = spectr_arr_2[k].α
        _λ = spectr_arr_2[k].λ
        _ε = arr_ε[k]

        def _fun_integrate(_x):
            return _Ψ(_x) * (_Xm(_x)*_Ψ(_x))

        _integrate, err = integrate.quad(_fun_integrate, a_left,
b_right)
        _sum += (1/_α) * (_λ - _ε) * _integrate

    _g.append(_sum)
Gm = _g
```

Данный код соответствует пункту 2-3 в алгоритме решения обратной задачи.

Определим функцию для исправления подобранной функции $u(x)$, собственных чисел λ и нормировочных коэффициентов α. В данном коде задана функция Хэвисайда, revers_e – параметр определяющий скорость градиентного метода.

```
def δU(_x):
    _sum = 0
    for m in range(1, M):
        _rangeX = [arrayX[m-1], arrayX[m]]
        _left = _rangeX[0]
        _right = _rangeX[1]

        def _Xm(_x):
            if (_left  <= _x <= _right):
                return 1
            else:
                return 0
        _sum += Gm[m-1] * _Xm(_x)

    return -revers_e *_sum
```

Данный код соответствует формуле (33) в пункте 4 алгоритма решения обратной задачи.

Приведём код для коррекции подобранной функции $u(x)$. В цикле корректируется каждый отрезок функции $u(x)$.

```
        for i in range(0, len(arrayU)):
            arrayU[i] = arrayU[i] + δU(arrayX[i])
```

Данный код соответствует формуле (32) в пункте 4 решении обратной задачи.

Приведём фрагмент кода, корректирующий собственные числа и нормировочные коэффициенты. В данном коде задан цикл для корректировки нормировочного коэффициента согласно формуле (36). Вычисляется невязка со скорректированными значениями собственных чисел и нормировочных коэффициентов согласно формуле (38). Для вычисления интеграла используется вспомогательная функция _funIntegr_δλ.

```
for i in range(N):
    _Ψ = spectr_arr_2[i].Ψ
    _α = spectr_arr_2[i].α
    _λ = spectr_arr_2[i].λ
    def _funIntegr_δλ(_x):
        return _Ψ(_x) * (δU(_x)*_Ψ(_x))

    _integr, err = integrate.quad(_funIntegr_δλ, a_left, b_right)
    _δλ = (1/_α)*(_integr)
    _λ = _λ + _δλ
    spectr_arr_2[i].λ = _λ
    _sumA = 0
    for k in range(N):
        if (i != k):
            _Ψk = spectr_arr_2[k].Ψ
            _αk = spectr_arr_2[k].α
            _λk = spectr_arr_2[k].λ
            def _funIntegr_aik(_x):
                return _Ψk(_x) * (U(_x)*_Ψ(_x))
            aik, err = integrate.quad(_funIntegr_aik, a_left, b_right)

            aik = aik/(_αk*(_λ-_λk))
            _sumA += _α * abs(aik)**2

    _α = _α + _sumA
    spectr_arr_2[i].α = _α
    _ε = arr_ε[i]
    _β = arr_β[i]
    _sum_λ += (_λ - _ε)**2
    _sum_α += (_α - _β)**2

delta_Δ = 1/2*(_sum_λ+_sum_α)
arr_Δ.append(delta_Δ)
```

Данный код соответствует пунктам 5, 6, 7 в алгоритме решения обратной задачи.

Таким образом реализован алгоритм решения обратной задачи Штурма-Лиувилля на интервале. Во время выполнения программы в консоль приложения выводятся вспомогательные данные, после всех вычислений приложение выводит в отдельных окнах график искомой функции $u^*(x)$, график подобранной функции $u(x)$, график зависимости собственных чисел от параметра стрельбы, окно, показывающее изменение подобранной функции $u(x)$ на каждом цикле корректировки, окно с графиком функции

$u^*(x)$ и графиком подобранной функции $u(x)$ на первом и на последнем шаге.

2.5 Численные эксперименты с готовым кодом

Решим обратную задачу Штурма-Лиувилля с помощью программного кода при условии: неизвестная функция на отрезке $[0; \pi]$ – парабола с минимумом равным минус 0.7, $u^*(x) = 0.7 \times \left(\left(\frac{2x-\pi}{\pi}\right)^2 - 1\right)$. Подобранная функция $u(x) = 0.9 \times \left(\left(\frac{2x-\pi}{\pi}\right)^2 - 1\right)$ – парабола с минимумом равным минус 0.9. Количество собственных чисел и нормировочных коэффициентов равно восьми. В таблице 2 показаны спектральные данные $(\xi_1, \beta_1, \dots, \xi_n, \beta_n)$ для функции $u^*(x)$ и спектральные данные $(\lambda_1, \alpha_1, \dots \lambda_n, \alpha_n)$ для функции $u(x)$.

Таблица 2 – данные рассеяния функции $u^*(x)$ *и* $u(x)$

	$u^*(x)$		$u(x)$	
№	ε	β	λ	α
1	-0.2041	1.3361	-0.2778	1.2521
2	0.6697	1.8224	0.6052	1.6995
3	3.5754	1.6668	3.4563	1.6621
4	8.5445	1.6109	8.4147	1.6049
5	15.5220	1.6051	15.3854	1.6044
6	24.5514	1.5913	24.4229	1.5896
7	35.5720	1.5769	35.4504	1.5763
8	48.5564	1.5769	48.4298	1.5755

В алгоритме решения обратной задачи вычисляется невязка, приведем результат вычисления невязки на каждой итерации в таблице 3. Видно, что невязка с каждой итерации уменьшается.

Таблица 3 – невязка на каждой итерации

№	Невязка
1	0.036751
2	0.020067

3	0.01249
4	0.009419

Результат выполнение программы показан на рисунке 9. По графику видно, что минимум подобранной функции уменьшился и подобранная функция приблизилась к первоначальной функции.

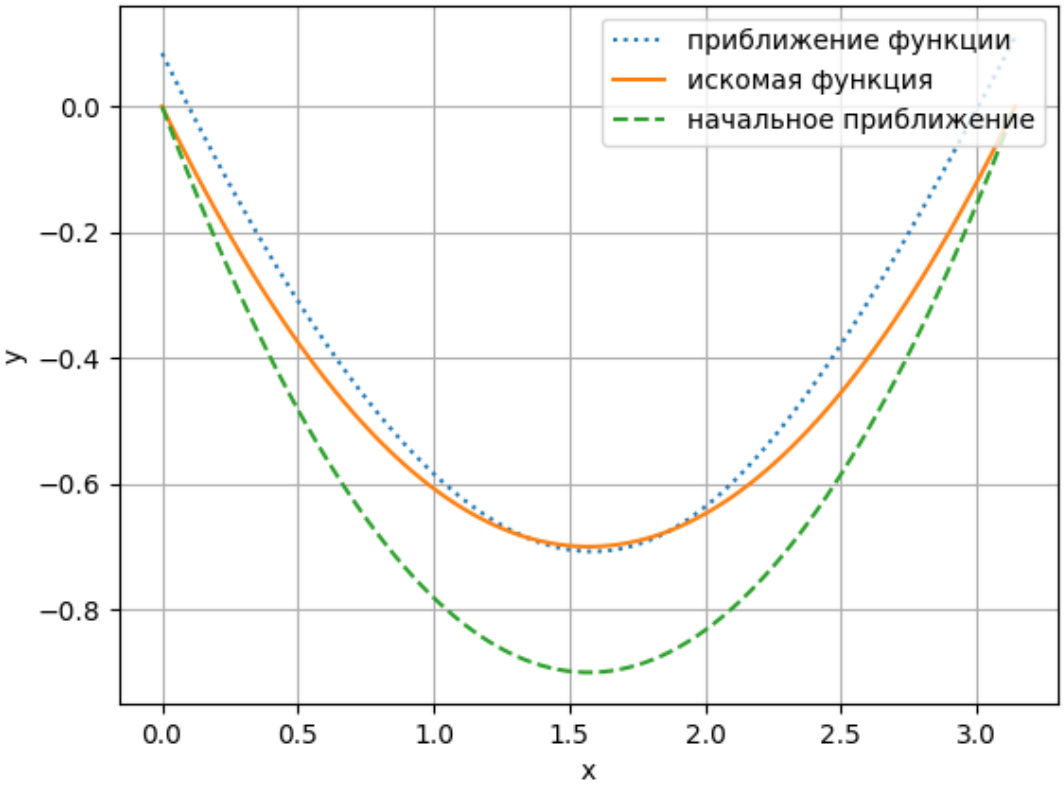

Рисунок 9 – Результат решения обратной задачи

На рисунке 10 показано как восстанавливалась подобранная функция $u(x)$ на каждой итерации. По рисунку видно, что нулевая итерация – подобранная функция $u(x)$, последняя итерация – приближение к первоначальной функции $u^*(x)$.

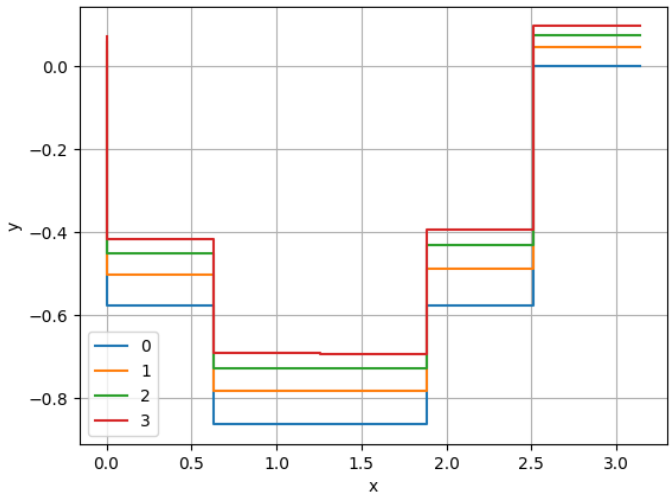

Рисунок 10 – Восстановление неизвестной функции

Восстановим более сложную функцию $u(x) = -\frac{1}{2}(2x - \pi) \times sin(\frac{(2x-\pi)^2}{2})$, аппроксимация этой функции представлена на рисунке 11 в виде ступенчатой функции.

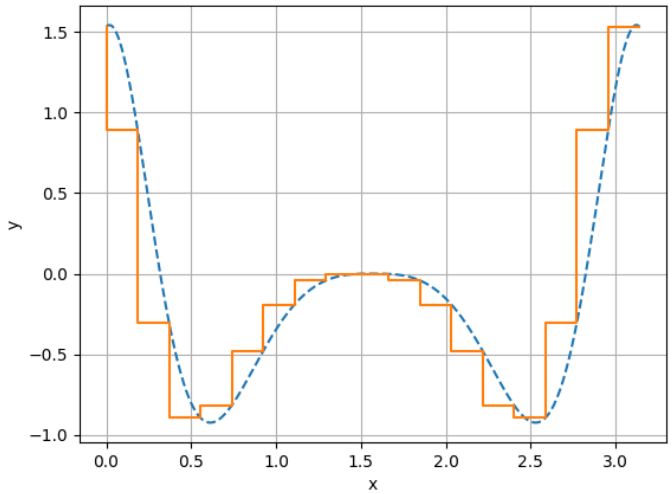

Рисунок 11 – Аппроксимация w-образной функции

Для восстановление исходной функции необходимо подобрать такую функцию $u(x)$, чтобы собственные значение этой функции были на одном уровне с собственными значениями w-образной функции $u^*(x)$. В качестве

подобранной функции возьмём параболу с минимумом в точке $y = 0$ и с границами совпадающими с w-образной функцией, график подобранной функции $u(x)$ показан на рисунке 12.

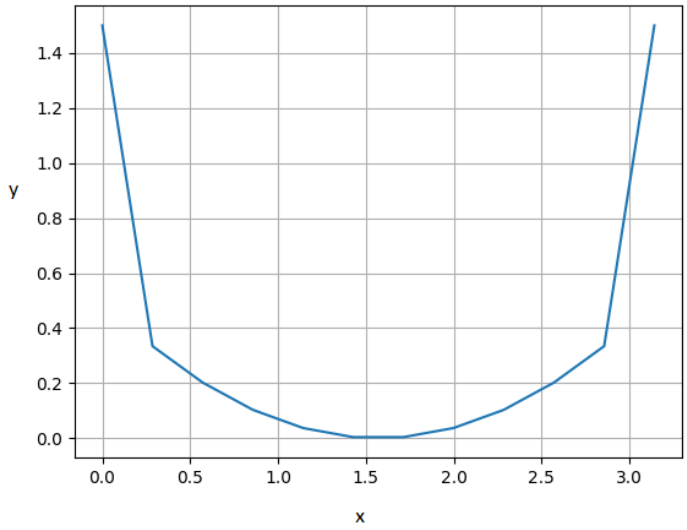

Рисунок 12 – Подобранная функция

Приведем в таблице 4 спектральные данные функции $u^*(x)$ и $u(x)$.

Таблица 4 – данные рассеяния функции $u^*(x)$ *и* $u(x)$

	$u^*(x)$		$u(x)$	
№	ε	β	λ	α
1	-0.2041	1.3361	-0.2778	1.2521
2	0.6697	1.8224	0.6052	1.6995
3	3.5754	1.6668	3.4563	1.6621
4	8.5445	1.6109	8.4147	1.6049
5	15.5220	1.6051	15.3854	1.6044
6	24.5514	1.5913	24.4229	1.5896
7	35.5720	1.5769	35.4504	1.5763
8	48.5564	1.5769	48.4298	1.5755
9	63.5482	1.5717	63.4196	1.5706

Результат работы программы показан на рисунке 13. Начальное приближение после корректировки приняла форму искомой функции, но не совпадает в местах изгиба с этой функцией.

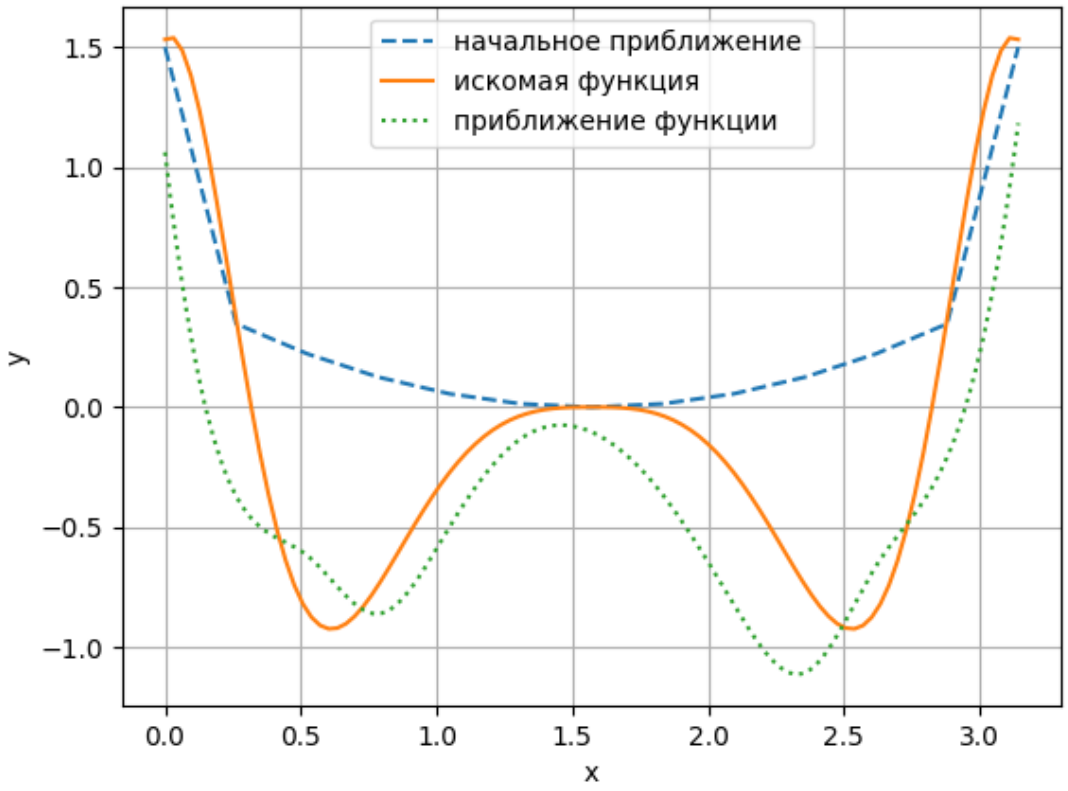

Рисунок 13 – Результат решения обратной задачи

Корректировка подобранной функции $u(x)$ на каждой итерации показана на рисунке 14. Видно, что с каждой итерации подобранная функция $u(x)$ приближается к исходной функции $u^*(x)$.

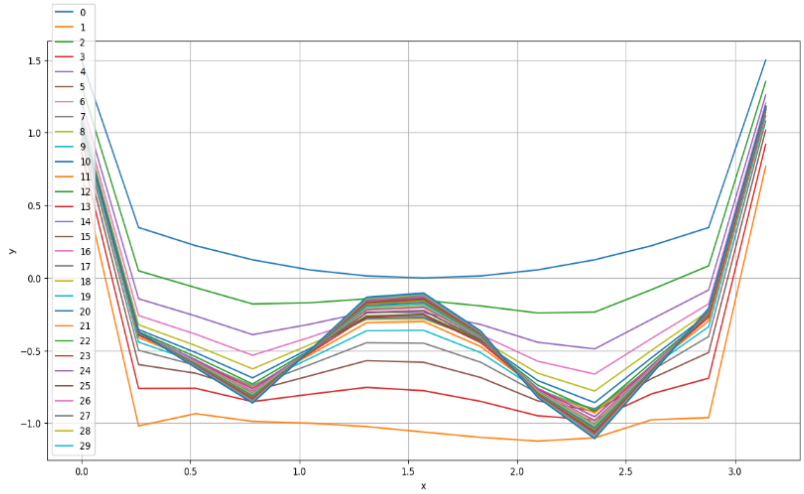

Рисунок 14 – Корректировка подобранной функции

Приведем результаты восстановления w-образной функции с различным количеством спектральных данных (N). На рисунке 15 показан результат восстановления при $N = 3$, на рисунке 16 при $N = 5$, на рисунке 17 при $N = 9$. Видно, что между $N = 3$ и $N = 5$ существенное различие, а между $N = 5$ и $N = 9$ различия минимальны.

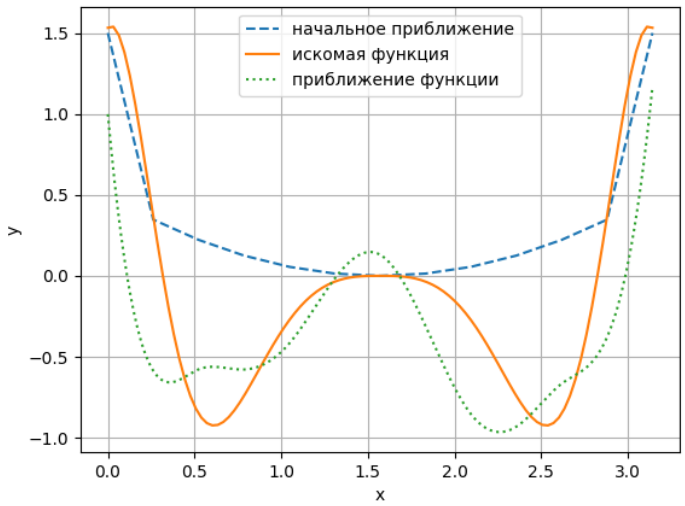

Рисунок 15 – Результат восстановления при N = 3

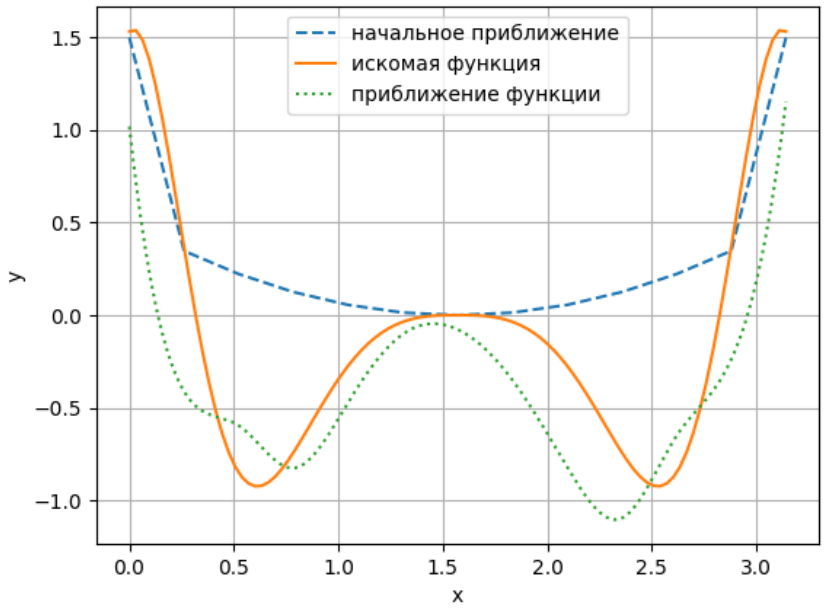

Рисунок 16 – Результат восстановления при N = 5

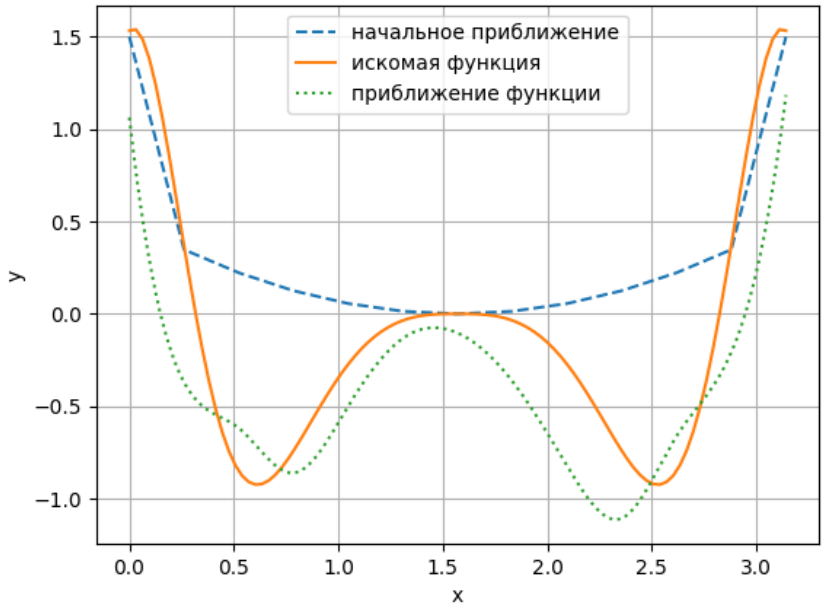

Рисунок 17 – Результат восстановления при N = 9

Приведем результаты восстановления w-образной функции с различными начальными приближениями на рисунке 18. По рисунку видно,

что начальное приближение сильно влияет на результат восстановления первоначальной функции.

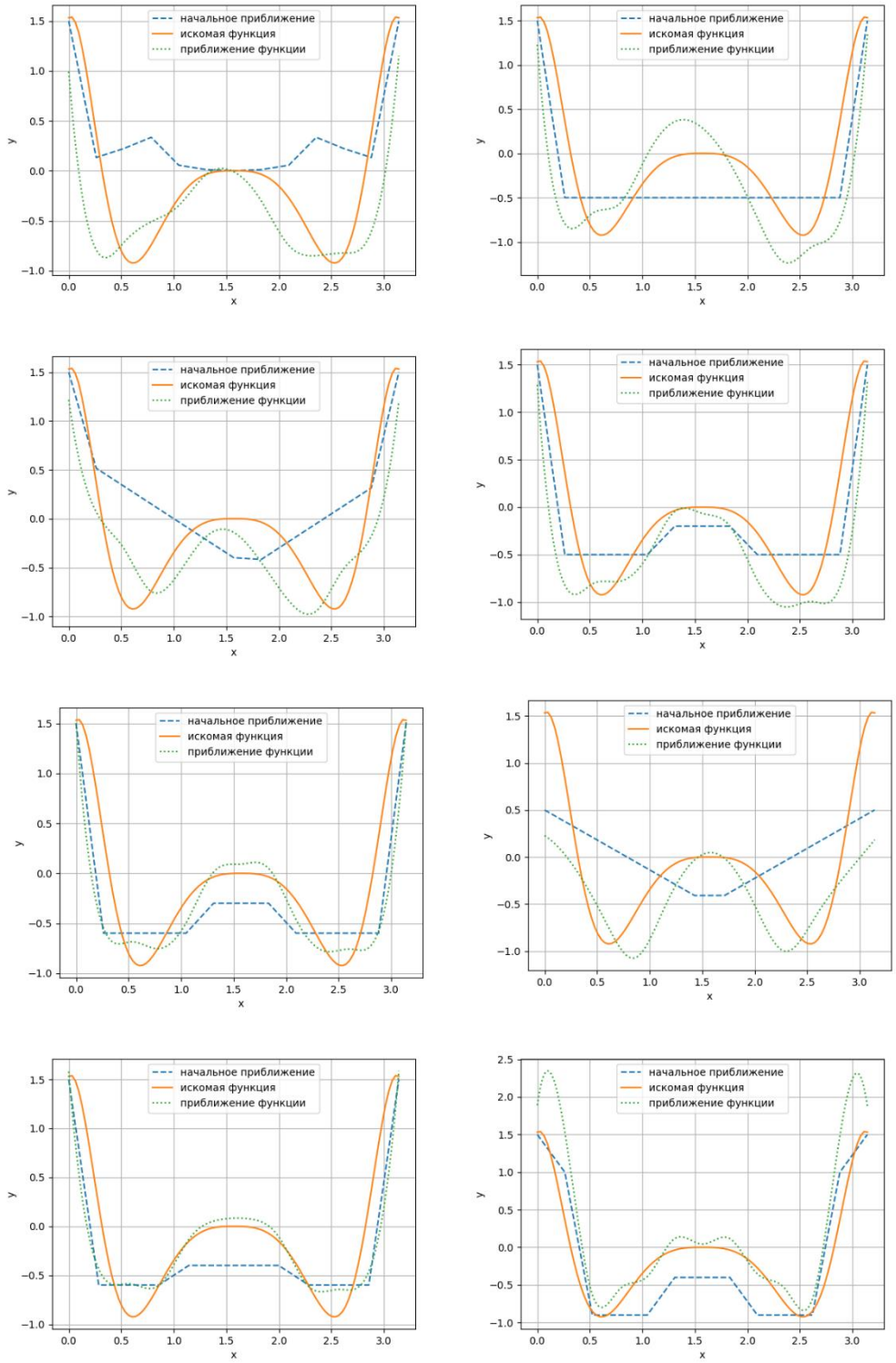

Рисунок 18 – Различные начальные приближения

Для исследования градиентного метода решения обратной задачи Штурма-Лиувилля на интервале задавалось различное количество спектральных данных при одинаковом начальном приближении, результат приводился в виде графиков. В результате исследования стало понятно, что после определенного количества спектральных данных влияние количества спектральных данных на результат не значительно. Так же было проведено исследование влияния начального приближения на результат восстановления первоначальной функции. Стало понятно, что от начального приближения сильно зависит результат восстановления первоначальной функции.

2.6 Вывод по второму разделу

Во втором разделе описана среда в которой проводились исследования, приведены и описаны все необходимые библиотеки языка Python для реализации программы, реализован алгоритм аппроксимации функции, алгоритм решения прямой и обратной задачи Штурма-Лиувилля на интервале. При описании реализации алгоритмов приведены фрагменты кода программы и приведены блок схемы самих алгоритмов. После реализации алгоритма градиентного метода решения обратной задачи Штурма-Лиувилля на интервале проведены серии численных экспериментов. По результатам исследований с готовой программой приведены соответствующие выводы.

Заключение

В результате выполнения выпускной квалификационной работы были проделаны все шаги для реализации и исследования градиентного метода решения обратной задачи Штурма-Лиувилля на интервале.

– подобраны инструменты и технологии для разработки программы;

– описан и запрограммирован алгоритм для решения прямой задачи Штурма-Лиувилля методом стрельбы;

– описан и запрограммирован градиентный метод решения обратной задачи Штурма-Лиувилля;

– проведены численные эксперименты с готовой программой.

В первом разделе сформулирована математическая модель задачи Штурма-Лиувилля. В подразделе 1.1 приведена вся необходимая литература для понимания ВКР. В подразделе 1.2 описан планарный волновод, показана схема распространения волн в волноводе и сформулирована математическая модель в общем виде на примере планарного волновода. В подразделе 1.3 сформулирована математическая модель прямой задачи, описан алгоритм аппроксимации исходной функции и описан алгоритм решения прямой задачи методом стрельбы. В подразделе 1.4 сформулирована математическая модель обратной задачи и описан алгоритм решения обратной задачи с помощью градиентного метода. Во втором разделе реализован алгоритм решения обратной задачи Штурма-Лиувилля. В подразделе 2.1 описаны используемые библиотеки для реализации программы на языке Python. В подразделе 2.2 описана реализация алгоритма аппроксимации исходной функции. В подразделе 2.3 реализован алгоритм метода стрельбы. В подразделе 2.4 реализован алгоритм градиентного метода для решения обратной задачи Штурма-Лиувилля на интервале. В подразделе 2.5 проведены численные эксперименты с готовым кодом.

Список используемой литературы

1. Алифанов О. М., Артюхин Е. А. Экстремальные методы решения некорректных задач и их приложения к обратным задачам теплообмена. – " Наука," Глав. ред. физико-математической лит-ры, 1988. – 286 с.

2. Бакушинский, А. Б. К распространению принципа невязки. – Журнал вычислительной математики и математической физики, 10(1), 210-213.

3. Григорьев Л.В. Кремниевая фотоника. Учебно-методическое пособие по практическим работам. – СПб: Университет ИТМО, 2015. – 69 с

4. Егоров А.А. Обратная задача рассеяния лазерного излучения в интегрально-оптическом волноводе с двумерными нерегулярностями при наличии шума. – Сборник научных трудов МНТОРЭС им. А.С. Попова «Интегральная оптика и волноводная оптоэлектроника», 2015. – 30 с.

5. Емелин И. В., Красносельский М. А. Правило останова в итерационных процедурах решения некорректных задач //Автоматика и телемеханика. – 1978. – №. 12. – С. 59-63.

6. Зеленовский П. С. Основы интегральной и волоконной оптики: учебное пособие. Учебное пособие. — Екатеринбург.: Издательство Уральского университета, 2019 – 136 с.

7. Кабанихин С. И. Обратные и некорректные задачи. Учебник для студентов высших учебных заведений. – Новосибирск: Сибирское научное издательство, 2009. – 457 с.

8. Кабанов С. Н. и др. Метод обратной задачи в теории нелинейных волн. Учебное пособие. – Саратовский государственный университет, 2013. – 115 с.

9. Карчевский М. М. Лекции по уравнениям математической физики. Учебное пособие. — 2-е изд., испр. — СПб.: Издательство «Лань», 2016. — 164 с.

10. Левитан Б. М. Обратные задачи Штурма-Лиувилля. – Наука, Главная редакция физико-математической литературы, 1984. – 240 с.

11. Марченко В. А. Операторы Штурма-Лиувилля и их приложения. – Наук. думка, 1975. – 330 с.

12. Морозов В.А. О регуляризирующих семействах операторов. – Изд-во Моск. ун-та Москва, 1967. – 93 с.

13. Никоноров Н. В., Шандаров С. М. Волноводная фотоника. Учебное пособие. — Санкт Петербург, ИТМО, 2008. – 142 с.

14. Поляк Б. Т. Градиентные методы минимизации функционалов //Журнал вычислительной математики и математической физики. – 1963. – Т. 3. – №. 4. – С. 643-653.

15. Самарский А. А., Вабищевич П. Н. Численные методы решения обратных задач математической физики. – Издательство ЛКИ, 2009. – 480 с.

16. Талалов С.В. Обратные и некорректные задачи. Электронное учебное пособие. ТГУ, 2019. – 61 с.

17. Талалов. С.В. Об одном варианте решения обратной задачи Штурма – Лиувилля градиентными методами. В сборнике: «Материалы IV всероссийской научной конференции с межд. участием «Информационные технологии в моделировании и управлении: подходы, методы, решения». Тольятти, 20 апреля 2022 г. ТГУ, 2022.

18. Тихонов А. Н., Арсенин В. Я. Методы решения некорректных задач. – М.: Наука, 1979. 9. Чередниченко ВГ, 1979. – 283 с.

19. Фаддеев Л. Д. Обратная задача квантовой теории рассеяния //Успехи математических наук. – 1959. – Т. 14. – №. 4 (88. – С. 57-119.

20. Чудов Л. А. Обратная задача Штурма-Лиувилля, Матем. сб., 1949, том 25(67), номер 3, 451–456

21. Açil M., Konuralp A. Reconstruction of potential function in inverse Sturm-Liouville problem via partial data. – An International Journal of Optimization and Control: Theories & Applications (IJOCTA). – 2021. – Т. 11. – №. 2. – С. 186-198.

22. Brown B. M. et al. A Numerical Procedure for the Inverse Sturm-Liouville Operator. – Factorization, Singular Operators and Related Problems. – Springer, Dordrecht, 2003. – С. 55-64.

23. Freiling G., Yurko V. A. Inverse Sturm-Liouville problems and their applications. – Huntington : NOVA Science Publishers, 2001.

24. Ledoux V. Study of special algorithms for solving Sturm-Liouville and Schrodinger equations : дис. – Ghent University, 2007. – 232 с.

25. Röhrl N. A least-squares functional for solving inverse Sturm–Liouville problems. – Inverse Problems. – 2005. – Т. 21. – №. 6. – С. 2009.

Приложение А

Планарные волноводы

«Планарный волновод представляет из себя набор из трех слоев с разным показателем преломления. На рисунке 1 волноводный слой отмечен голубым цветом, он ограничен снизу подложкой и сверху покровными слоем. Волноводный слой имеет показатель преломления n_1 и толщину h, подложка имеет степень преломления n_2, ограничена сверху волноводным слоем и не ограничена снизу в направлении $-x$, покровный слой имеет степень преломления n_3, ограничен снизу волноводным слоем и не ограничен сверху в направлении x» [6], [13].

Планарные волноводы делятся на пленочные и градиентные. Плёночный планарный волновод представляет из себя пленку, нанесенную на подложку с меньшим показателем преломления. Пленка представляет из себя однородный материал с постоянной толщиной. Градиентные планарные волноводы имеют не однородный волноводный слой. Показатель преломления волноводного слоя плавно меняется вдоль координаты x, при этом больший показатель преломления на поверхности волноводного слоя. Рассмотрим пленочный планарный волновод:

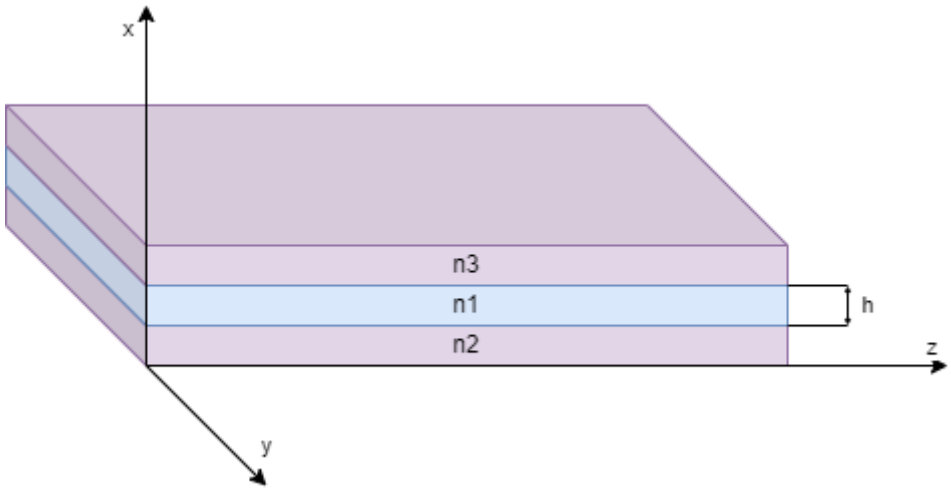

Рисунок 1 – Трехслойный планарный волновод

«Согласно электромагнитной теории электрические и магнитные явления можно описать четырьмя уравнениями Максвелла:

$$\mathrm{div}\overrightarrow{D} = \rho, \quad (1)$$

$$\mathrm{div}\overrightarrow{B} = 0, \quad (2)$$

$$\mathrm{rot}\overrightarrow{H} = \vec{j} + \frac{\partial \overrightarrow{D}}{\partial t}, \quad (3)$$

$$\mathrm{rot}\overrightarrow{H} = -\frac{\partial \overrightarrow{B}}{\partial t}, \quad (4)$$

где $\vec{D}$ — вектор электрической индукции;

$\vec{B}$ — вектор магнитной индукции;

$\vec{E}$ — напряженность электрического поля;

$\vec{D}$ — напряженность магнитного поля;

$\vec{j}$ — плотность тока,

ε— диэлектрическая проницаемость;

ε_0 — диэлектрическая постоянная;

μ — магнитная проницаемость;

μ_0 — магнитная постоянная;

σ — удельная проводимость;

ρ — объемная плотность заряда.» [6].

Добавим к уравнениям материальные соотношения.

$$\vec{D} = \varepsilon\varepsilon_0\vec{H},\ \vec{B} = \mu\mu_0\vec{H},\ \vec{j} = \sigma\vec{E}. \quad (5)$$

«В случае распространения электромагнитной волны в однородной среде(($\sigma = 0$, $\rho = 0$, $\vec{j} = 0$) с постоянными ε и μ уравнение можно записать в таком виде:

$$\mathrm{div}\overrightarrow{\mathrm{E}} = 0, \tag{6}$$

$$\mathrm{div}\overrightarrow{\mathrm{H}} = 0, \tag{7}$$

$$\mathrm{rot}\overrightarrow{\mathrm{H}} = \varepsilon\varepsilon_0 \frac{\partial \overrightarrow{\mathrm{E}}}{\partial \mathrm{t}}, \tag{8}$$

$$\mathrm{rot}\overrightarrow{\mathrm{E}} = -\mu\mu_0 \frac{\partial \overrightarrow{\mathrm{H}}}{\partial \mathrm{t}}. \tag{9}$$

Применим к обеим частям уравнения операцию rot:

$$\mathrm{rot}\,\mathrm{rot}\overrightarrow{\mathrm{E}} = -\mu\mu_0 \frac{\partial\,(\mathrm{rot}\,\overrightarrow{\mathrm{H}})}{\partial \mathrm{t}} = -\varepsilon\varepsilon_0\mu\mu_0 \frac{\partial^2\overrightarrow{\mathrm{E}}}{\partial \mathrm{t}^2}. \tag{10}$$

Так как $\mathrm{rot}\,\mathrm{rot}\overrightarrow{\mathrm{E}} = \overrightarrow{\nabla} \times \left(\overrightarrow{\nabla} \times \overrightarrow{\mathrm{E}}\right) = \overrightarrow{\nabla} * \overrightarrow{\mathrm{E}} - \nabla^2\overrightarrow{\mathrm{E}} = -\nabla^2\overrightarrow{\mathrm{E}}$, то получаем волновое уравнение для напряженности электрического поля:

$$\nabla^2\overrightarrow{\mathrm{E}} - \varepsilon\varepsilon_0\mu\mu_0 \frac{\partial^2\overrightarrow{\mathrm{E}}}{\partial \mathrm{t}^2} = 0. \tag{11}$$

Так же можем получить волновое уравнение для напряженности магнитного поля:

$$\nabla^2\overrightarrow{\mathrm{E}} - \varepsilon\varepsilon_0\mu\mu_0 \frac{\partial^2\overrightarrow{\mathrm{H}}}{\partial \mathrm{t}^2} = 0. \tag{12}$$

Таким образом получаются волновые уравнения для электрического и магнитного поля.» [6].

Рассмотрим планарный волновод с показателем преломления n_1 и окруженный материалами с меньшим показателем преломления n_2 и n_3. Световая волна распространяется только вдоль одной оси z, а ось x перпендикулярная плоскости волновода. «Решение волнового уравнения (11) – плоская монохроматическая волна с напряженностями электрического и магнитного поля.

$$\overrightarrow{E} = \overrightarrow{E}_0(x,y)\exp\big(i(\omega t - \beta z)\big), \quad (13)$$

$$\overrightarrow{H} = \overrightarrow{H}_0(x,y)\exp(i(\omega t - \beta z)), \quad (14)$$

где ω — угловая частота;

β — постоянная распространения волны вдоль направления Z, которая зависит от граничных условий, накладываемых на $\overrightarrow{E}_0(x,y)$.» [11].

Для волн перпендикулярных плоскости падения:

$$\beta E_y = \mu_0 \omega H_x, \quad (15)$$

$$-i\beta E_x - \frac{\partial H_z}{\partial x} = i\varepsilon\varepsilon_0\omega E_y, \quad (16)$$

$$\frac{\partial E_y}{\partial x} = -i\mu_0\omega H_z. \quad (17)$$

Выразим из уравнения (15) H_x, а из уравнения (17) H_z и подставим в уравнение (16). После подстановки получим волновое уравнение для E_y в виде:

$$\frac{\partial^2 E_y}{\partial x^2} + (\omega^2\varepsilon\varepsilon_0\mu_0 - \beta^2)E_y = 0. \quad (18)$$

Решение для этого уравнение - монохроматические плоские волны, распространяющиеся вдоль направления x:

$$E_y = E_{y0}\exp(ik_x x), \quad (19)$$

где k_x – волновой вектор распространения волны в направлении x;

n – показатель преломления.

Если сделать замену $k^2n^2 - \beta^2 = \omega^2\varepsilon\varepsilon_0\mu_0 - \beta^2$, то получим уравнение Штурма—Лиувилля:

$$\frac{\partial^2 E_y}{\partial x^2} + (k^2n^2 - \beta^2)E_y = 0. \quad (20)$$

Таким образом можно свести уравнение Максвелла к уравнению Штурма—Лиувилля.

Приложение Б

Код программы

```
# подключаем библиотеки
from ast import Delete
from cProfile import label
import math
import string
import sys
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
from sympy.core.symbol import symbols

from scipy.interpolate import CubicSpline
from scipy.optimize import fsolve
import sympy as sym
import random
import warnings


# задаем константы
π = math.pi
Q0 = 0
Qπ = 0
M = 2
a_left = 0
b_right = π

# первоночальная функция
def fun_0_π(x):
# 1
    X = (x*2-π)
    return -(1/2)*abs(X)*np.sin((X**2)/2)


# подобранная функция
def fun_0_π_2(x):
    X = (x*2-π)/π
    return 0.5*((X**2)/1 )
```

Продолжение Приложения Б

```
# функция для создания отрезков
def createFirstU(fun):
    arrX = np.linspace(a_left, b_right, M)
    U = []
    for x in arrX:
        U.append(fun(x))

    U = [1.5] + U[1:-1] + [1.5]
    return U

# определение знака
def sign(x):
    if x >= 0:
        return 1
    else:
        return 0

# класс для хранения данных рассеяния
class elem_spectr:
    def __init__(self, λ, α, Ψ, A, B):
        """Constructor"""
        self.λ = λ
        self.α = α
        self.Ψ = Ψ
        self.A = A
        self.B = B


# метод стрельбы
def shoot(fun, delta, e, eps_shoot, a, b, m, Δ, step):
    print("метод стрельбы")
    fig, ax = plt.subplots()
    ax.grid()
    x = np.linspace(a, b, 1000)
    y = np.vectorize(fun, otypes=[float])
    plt.plot(x, y(x), label="искомая функция", linestyle='dashed')
    plt.xlabel("x")
    plt.ylabel("y")


# начало цикла
    while delta > e:
        # функция для нахождения нормы
        def f1(x):
            return pow((abs(fun(x)-fun(Ej))), 2)
```

```
        def f2(x):
            return pow((abs(fun(x))), 2)
        # делим отрезок [a:b] на m частей
        xx = np.linspace(a, b, m)
# вычисляем расстояние между соседними точками
        h = xx[1]-xx[0]
        fx = y(xx)
        xjminus1 = xx[0]
        sum1 = 0
#   находим
        sum2, err = integrate.quad(f2, a, b)

        for x in xx[1:]:
            # вычисляем середины соседних отрезков
            Ej = (xjminus1 + x)/2
# v1 для вычисления нормы
            v1, err = integrate.quad(f1, xjminus1, x)
            sum1 += v1
            xjminus1 = x
# вычисляем величину дельта, для условия выхода из цикла
        delta1 = pow(sum1, 1/2)
        delta2 = pow(sum2, 1/2)
        delta = delta1/delta2
        m = m+1

# строим аппроксимацию
    if step:
        plt.step(xx, fx)

    plt.xlabel("x")
    plt.ylabel("y")
    m = len(xx)
    print("число разбиений отрезка M = ", m)

# начальные данные
    x = xx
    u = fx
    spectr = []

    u0 = min(u)
    u = [abs(t) for t in u]
```

```
    λ_graph = []
    a = []
    α = []

    print("минимум функции u0 =", u0)
    λ = u0 + 0.0001

    while True:
        array_Λ = []
        array_A = []
        array_B = []
        array_α = []

        for j in range(1, m):
            Λ = u[j] + λ
            sqrt_Λ = np.sqrt(abs(Λ))
            j_1 = j - 1
            A, B, X = symbols('A, B, X')

            if Λ > 0:
                funΨ = A * sym.cos(sqrt_Λ * X) + B * sym.sin(sqrt_Λ * X)
                def Ψ(x):
                    return A * sym.cos(sqrt_Λ * x) + B * sym.sin(sqrt_Λ * x)
            if Λ < 0:
                funΨ = A * sym.cosh(sqrt_Λ * X) + B * sym.sinh(sqrt_Λ * X)
                def Ψ(x):
                    return A * sym.cosh(sqrt_Λ * x) + B * sym.sinh(sqrt_Λ * x)

# Начальные коэффициенты
            if j == 1:
                A, B, X = symbols('A, B, X')

                funΨ = funΨ - 1
                funΨdx = sym.diff(funΨ, X)
                funΨdx = funΨdx + Q0

                funΨ = funΨ.subs(X, 0)
                funΨdx = funΨdx.subs(X, 0)

                A, B = list(sym.solve([funΨ,  funΨdx], A, B).values())
```

```
# Сшивание
            else:
                sqrt_Λ_minus1 = np.sqrt(abs(array_Λ[-1]))
                A, B = symbols('A, B')

                if array_Λ[-1] > 0 and Λ > 0:
                    A, B = list(sym.solve([array_A[-1] * sym.cos(sqrt_Λ_minus1
* x[j_1]) + array_B[-1] * sym.sin(sqrt_Λ_minus1 * x[j_1]) - A * sym.cos(sqrt_Λ
* x[j_1]) - B * sym.sin(sqrt_Λ * x[j_1]),
                                                -array_A[-1] * sqrt_Λ_minus1 *
sym.sin(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cos(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sin(sqrt_Λ * x[j_1]) - B *
sqrt_Λ * sym.cos(sqrt_Λ * x[j_1])], A, B).values())

                elif array_Λ[-1] > 0 and Λ < 0:
                    A, B = list(sym.solve([array_A[-1] * sym.cos(sqrt_Λ_minus1
* x[j_1]) + array_B[-1] * sym.sin(sqrt_Λ_minus1 * x[j_1]) - A *
sym.cosh(sqrt_Λ * x[j_1]) - B * sym.sinh(sqrt_Λ * x[j_1]),
                                                -array_A[-1] * sqrt_Λ_minus1 *
sym.sin(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cos(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sinh(sqrt_Λ * x[j_1]) - B *
sqrt_Λ * sym.cosh(sqrt_Λ * x[j_1])], A, B).values())

                elif array_Λ[-1] < 0 and Λ > 0:
                    A, B = list(sym.solve([array_A[-1] *
sym.cosh(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sym.sinh(sqrt_Λ_minus1 *
x[j_1]) - A * sym.cos(sqrt_Λ * x[j_1]) - B * sym.sin(sqrt_Λ * x[j_1]),
                                                -array_A[-1] * sqrt_Λ_minus1 *
sym.sinh(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cosh(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sin(sqrt_Λ * x[j_1]) - B *
sqrt_Λ * sym.cos(sqrt_Λ * x[j_1])], A, B).values())

                elif array_Λ[-1] < 0 and Λ < 0:
                    A, B = list(sym.solve([array_A[-1] *
sym.cosh(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sym.sinh(sqrt_Λ_minus1 *
x[j_1]) - A * sym.cosh(sqrt_Λ * x[j_1]) - B * sym.sinh(sqrt_Λ * x[j_1]),
                                                -array_A[-1] * sqrt_Λ_minus1 *
sym.sinh(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cosh(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sinh(sqrt_Λ * x[j_1]) - B
* sqrt_Λ * sym.cosh(sqrt_Λ * x[j_1])], A, B).values())
```

```
            array_Λ.append(Λ)
             array_A.append(A)
             array_B.append(B)

             def Ψ_2(x):
                 return Ψ(x)**2

             integr, err = integrate.quad(Ψ_2, x[j_1], x[j])
             array_α.append(integr)

         x_symbol = sym.Symbol('x')
         Ψdx = sym.diff(Ψ(x_symbol))
         a_ = float(Ψdx.subs(x_symbol, π) + Qπ * Ψ(π))
         a.append(a_)

         α_ = sum(array_α)
         α.append(α_)
         λ_graph.append(λ)


         if len(a) >= 2 and a[-2] != 0 and np.sign(a[-1]) != np.sign(a[-2]):
             λ_ = fsolve(lambda x: [(a[-2] - a[-1]) * x[0] + (λ_graph[-2] -
λ_graph[-1])
                                    * x[1] + (λ_graph[-2] * a[-1] - λ_graph[-
1] * a[-2]), x[1]], [1, 1])[0]

             round_N = 3
             λ_ = round(λ_, round_N)
             α_ = round(α_, round_N)
             spectr.append(elem_spectr(λ_, α_, funΨ, A, B ))
             print( (len(spectr)), ") ", "λ=", λ_, "α=", α_)

         # Проверка точности
         if len(spectr) > 0:
             diff = abs(
                 len(spectr)+1 - math.sqrt(abs(λ_graph[-1]))) /
math.sqrt((abs(λ_graph[-1])))

             if diff <= eps_shoot:
                 break

             if len(spectr) >= EPSN:
                 break
```

```
        λ += Δ

    return spectr, m, λ_graph, α, a


# обратная задача
def reverse(fun_reverse, count_iter,  arr_ε, arr_β, len_λ):
    arr_Δ = []
    Gm = []

    print('Обратная задача')
    arrayX = np.linspace(a_left, b_right, M)
    arrayU = createFirstU(fun_reverse)

    _spectr, _, _, _, _, = shoot(fun_reverse, delta, e, eps_shoot, a_left,
b_right, M, Δ, True)

    fig, ax = plt.subplots()
    ax.grid()

    arr_λ = []
    arr_α = []

    for i in range(len_λ):
        _Ψ = _spectr[i].Ψ
        _A = _spectr[i].A
        _B = _spectr[i].B
        A, B, X = symbols('A, B, X')
        _Ψ = _Ψ.subs(A, _A)
        _Ψ = _Ψ.subs(B, _B)
        _Ψ = sym.lambdify(X, _Ψ)
        _spectr[i].Ψ = _Ψ

        arr_λ.append(_spectr[i].λ)
        arr_α.append(_spectr[i].α)


    print("------------------------")
    print("ИСКОМАЯ ФУНКЦИЯ")
    print("собственные числа. ε =", arr_ε)
    print("нормировочные коэффициенты. β =", arr_β)
    print("------------------------")
```

```
print("НАЧАЛЬНОЕ ПРИБЛИЖЕНИЕ")
print("собственные числа. λ =", arr_λ)
print("нормировочные коэффициенты. α =", arr_α)
print("------------------------")

def U(_x):
    _sum = 0
    for m in range(1, M):
        _rangeX = [arrayX[m-1], arrayX[m]]
        _left = _rangeX[0]
        _right = _rangeX[1]

        def _Xm(_x):
            if (_left  <= _x <= _right):
                return 1
            else:
                return 0
        _sum += arrayU[m] * _Xm(_x)
    return _sum


for ccc in range(count_iter):
    ax.grid()
    # plt.step(arrayX, arrayU)
    plt.plot(arrayX, arrayU, label=(ccc))
    ax.grid()
    ax.legend(loc="lower left")
    plt.xlabel("x")
    plt.ylabel("y")

    diff = 0
    for i in range(len_λ):
        diff += abs(arr_λ[i]-arr_ε[i] )
        diff += abs(arr_α[i]-arr_β[i] )

#  вычисления градиента невязки
    _g = []
    for m in range(1, M):
        _sum = 0
        _rangeX = [arrayX[m-1], arrayX[m]]
        _left = _rangeX[0]
        _right = _rangeX[1]
```

```
            def _Xm(_x):
                if (_left  <= _x <= _right):
                    return 1
                else:
                    return 0

             for k in range(len_λ):
                _Ψ = _spectr[k].Ψ
                _α = arr_α[k]
                _λ = arr_λ[k]
                _ε = arr_ε[k]

                def _fun_integrate(_x):
                    return _Ψ(_x) * (_Xm(_x)*_Ψ(_x))

                _integrate, _ = integrate.quad(_fun_integrate, a_left,
b_right, limit=49,)
                _sum += (1/_α) * (_λ - _ε) * _integrate

            _g.append(_sum)

        Gm = _g

        def δU(_x):
            _sum = 0
            for m in range(1, M):
                _rangeX = [arrayX[m-1], arrayX[m]]
                _left = _rangeX[0]
                _right = _rangeX[1]

                def _Xm(_x):
                    if (_left  <= _x <= _right):
                        return 1
                    else:
                        return 0

                _sum += Gm[m-1] * _Xm(_x)
            return -revers_e *_sum
```

```
# корректировка подобранной функции
        arr_δU = []
        for i in range(0, len(arrayU)):
            arrayU[i] = arrayU[i] + δU(arrayX[i])
            arr_δU.append(δU(arrayX[i]))

#  корректировка собственных функций и нормировочного коэффициэнта
        _sum_λ = 0
        _sum_α = 0
        for i in range(len_λ):
            _Ψ = _spectr[i].Ψ
            _α = arr_α[i]
            _λ = arr_λ[i]

            def _funIntegr_δλ(_x):
                return _Ψ(_x) * (δU(_x)*_Ψ(_x))

            _integr, _ = integrate.quad(_funIntegr_δλ, a_left, b_right, lim-
it=50)

            _δλ = (1/_α)*(_integr)
            _λ = _λ + _δλ
            arr_λ[i] = _λ
            _sumA = 0
            for k in range(len_λ):
                if (i != k):

                    _Ψk = _spectr[k].Ψ
                    _αk = arr_α[k]
                    _λk = arr_λ[k]

                    def _funIntegr_aik(_x):
                        return _Ψk(_x) * (U(_x)*_Ψ(_x))
                    aik, err = integrate.quad(_funIntegr_aik, a_left, b_right,
limit=50)

                    aik = aik/(_αk*(_λ-_λk))
                    _sumA += _α * abs(aik)**2
```

Продолжение Приложения Б

```
        delta_Δ = 1/2*(_sum_λ+_sum_α)
        round_N = 3
        delta_Δ = round(delta_Δ, round_N)
        arr_Δ.append(delta_Δ)

    print("Δ", arr_Δ)
    plt.plot(arrayX, arrayU, label="last")
    return arrayX, arrayU


if __name__ == "__main__":
    warnings.simplefilter("ignore")
    print('------------------------------------------------------------------------
-----------------------------------------------')
    print('Студент:     Никитин Иван Сергеевич')
    print('Группа:      ПМИб-1802а')
    print('Тема ВКР:    «Реализация и исследование градиентных методов решения
обратной задачи Штурма-Лиувилля на интервале»')
    print('------------------------------------------------------------------------
-----------------------------------------------')

# начальные параметры

# параметры для метода стрельбы
    e = 35/100
    eps_shoot = 0/100
    EPSN = 9
    Δ = 0.7
    delta = 1

# параметры для обратной задачи

# параметр скорости спуска
    revers_e = 2.3
# число итераций приближения
    count_asd = 30


# получение спектра для начальной функции
    spectr = []
    spectr_arr_orig, M, λ_graph, α_graph, a_graph  = shoot(fun_0_π, delta, e,
eps_shoot, a_left, b_right, M, Δ, True)

    len_λ =  len(spectr_arr_orig)
```

```
# график a / λ
    a_int = CubicSpline(λ_graph, a_graph)
    λ_int = np.linspace(λ_graph[0], λ_graph[-1], num=1000, endpoint=True)
    fig, ax = plt.subplots()
    ax.set_ylim([-10, 10])
    ax.grid()
    plt.plot(λ_int, a_int(λ_int), color='m', )
    plt.ylabel('α(λ)')
    plt.xlabel('λ')

# найденные собственные числа
    _pred = a_int(0)
    _N_result = 0
    for i in range(1, len(λ_int)):
        _currX = λ_int[i]
        _currY = a_int(_currX)
        if sign(_currY) != sign(_pred):
            _N_result +=1
            plt.plot(_currX, 0,'ro')
        _pred = a_int(_currX)
    plt.show(block=False)


    arr_ε = []
    arr_β = []

    for i in range(len_λ):
        arr_ε.append(spectr_arr_orig[i].λ)
        arr_β.append(spectr_arr_orig[i].α )


#обратная задача
    arrayX, arrayU = reverse(fun_0_π_2, count_asd, arr_ε, arr_β, len_λ)
# построение графиков
    fig, ax_last = plt.subplots()
    ax_last.grid()
```

```
# подобранная функция
    plotX = np.linspace(a_left, b_right, M)
    plotY = createFirstU(fun_0_π_2)
    plt.plot(plotX, plotY, label="начальное приближение", lin-
estyle='dashed')

# начальная функция
    x = np.linspace(a_left, b_right, 1000)
    y = np.vectorize(fun_0_π, otypes=[float])
    plt.plot(x, y(x), label="искомая функция", linestyle='solid')


# приближение к начальной функции
    newFun =  CubicSpline(arrayX, arrayU)
    newFunX = np.linspace(a_left, b_right, num=1000, endpoint=True)
    plt.plot(newFunX, newFun(newFunX), label="приближение функции", lin-
estyle='dotted')
    ax_last.legend()
    plt.xlabel("x")
    plt.ylabel("y")

# конец

    plt.show()
    plt.close()
```

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

ОТЗЫВ
руководителя о выпускной квалификационной работе

Обучающийся Никитин Иван Сергеевич
(Фамилия Имя Отчество (при наличии) в именительном падеже)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ПМИб-1802а
(шифр группы)

Тема «Реализация и исследование градиентных методов решения обратной задачи Штурма-Лиувилля на интервале»

Перед студентом была поставлена задача составить алгоритм решения обратной задачи Штурма – Лиувилля на интервале градиентными методами, а также реализовать данный алгоритм на языке программирования Python 3.X. Необходимо было также провести ряд численных экспериментов с готовой программой, варьируя выбор начальной точки (функции). Результаты необходимо было вывести графически. Студент успешно справился с задачей. Алгоритм решения задачи градиентным методом составлен, программа написана и является работоспособной: по заданным спектральным характеристикам определяется коэффициентная функция («потенциал») уравнения Штурма – Лиувилля при заданных граничных условиях. При этом количество заданных точек спектра (которое в общем случае в данной задаче бесконечно) определялось исходя из заданной точности их определения при решении прямой задачи. В ходе выполнения выпускной квалификационной работы обучающийся освоил теорию решения прямой и обратной задачи Штурма – Лиувилля и успешно применил ее на практике.

К недостаткам выпускной квалификационной работы можно отнести неполный обзор литературы, а также малое количество выбранных начальных точек в численных экспериментах. Однако данные недостатки не снижают ценности работы. Рекомендованная оценка – «отлично».

профессор кафедры «Прикладная математика и информатика»
(должность, место работы полностью,

д.ф.-м.н., доцент _____ С.В. Талалов
ученая степень (при наличии), ученое звание (при наличии)) (подпись) (Инициалы Фамилия)

«21» июня 2022 г.